



Université Blida 1
Faculté des Sciences
Département d'Informatique



Master IL (Ingénierie Logiciel), SIR (Systèmes Informatiques et Réseaux) & SSI
(Sécurité des Systèmes d'Information)

Semestre 1

ALGORITHMIQUE AVANCÉE

M^{me} AROUSSI (s_roussi@esi.dz)

2020 - 2021

Disponible sur <https://sites.google.com/a/esi.dz/informatiqueblida/>

<https://elearning.univ-blida.dz/course/view.php?id=822>

PRÉAMBULE

Master	IL	SIR	SSI
Prérequis	Algo2-S3		
Volume horaire hebdomadaire	1.5H Cours + 1.5H TD		
Évaluation	Continu + Examen		
Note TD	10 points : les deux meilleures parties 10 points: devoirs, préparation, assiduité, présence, interrogation,		
Coefficient	2		3
Crédit	4		6

OBJECTIFS DU COURS

- Donner un panorama des structures et des méthodes que nous retrouvons dans divers domaines d'applications algorithmiques: réseaux, robotique, compilation, base de données, etc...
- Savoir analyser et comparer les performances de différentes solutions algorithmiques.

CONTENU DU COURS

- I. Arbres de Recherche
- II. NP-Complétude
- III. Heuristiques & Méta-heuristiques



Université Blida 1
Faculté des Sciences
Département d'Informatique



Master IL (Ingénierie Logiciel) & SIR (Systèmes Informatiques et Réseaux)
Semestre 1

CHAPITRE I:

ARBRES DE RECHERCHE

M^{me} AROUSSI (s_aroussi@esi.dz)

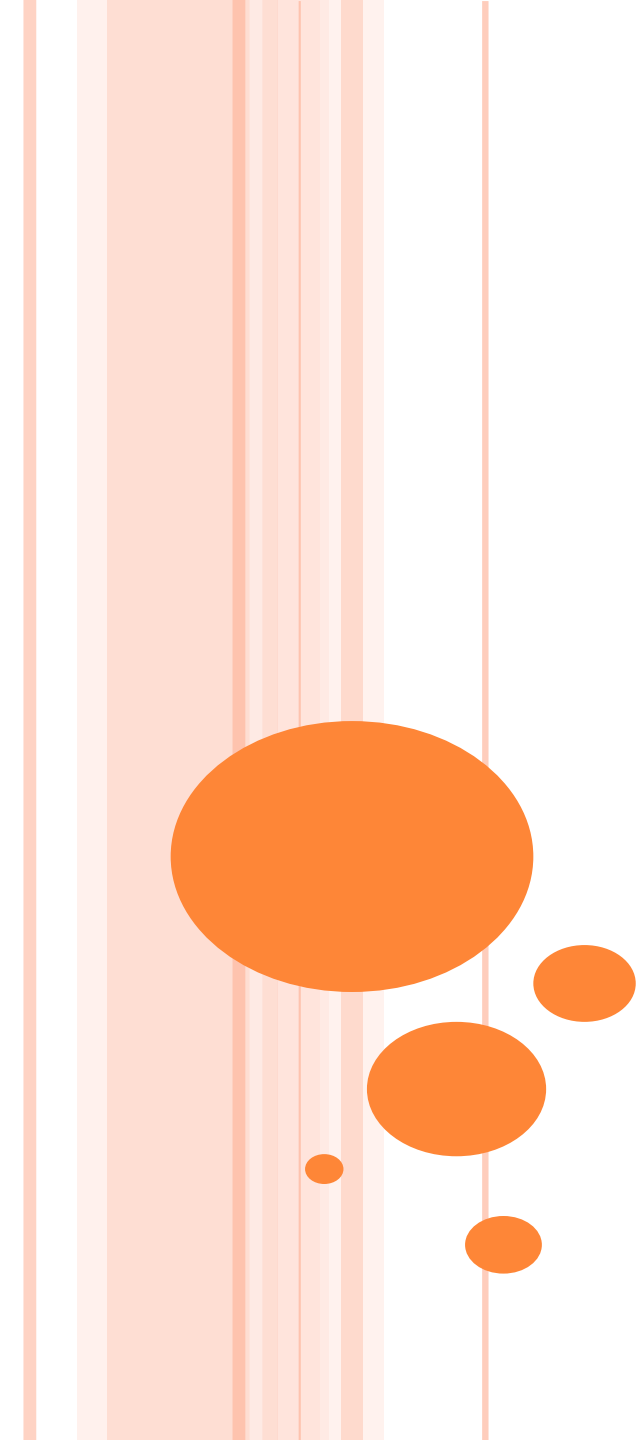
2020 - 2021

Disponible sur <https://sites.google.com/a/esi.dz/informatiqueblida/>

<https://elearning.univ-blida.dz/course/view.php?id=822>

PLAN DU CHAPITRE I

- ❑ Partie I: Introduction
- ❑ Partie II: Arbres Binaires de Recherche (**ABR**)
- ❑ Partie III: Arbres Binaires de Equilibrés (**AVL & TAS**)
- ❑ Partie IV: Arbres M-aire de Recherche (**AMR**)
- ❑ Partie V: Arbres M-aire de Recherche Equilibrés (**B-Arbres**)



PARTIE I: INTRODUCTION

PLAN DE LA PARTIE I

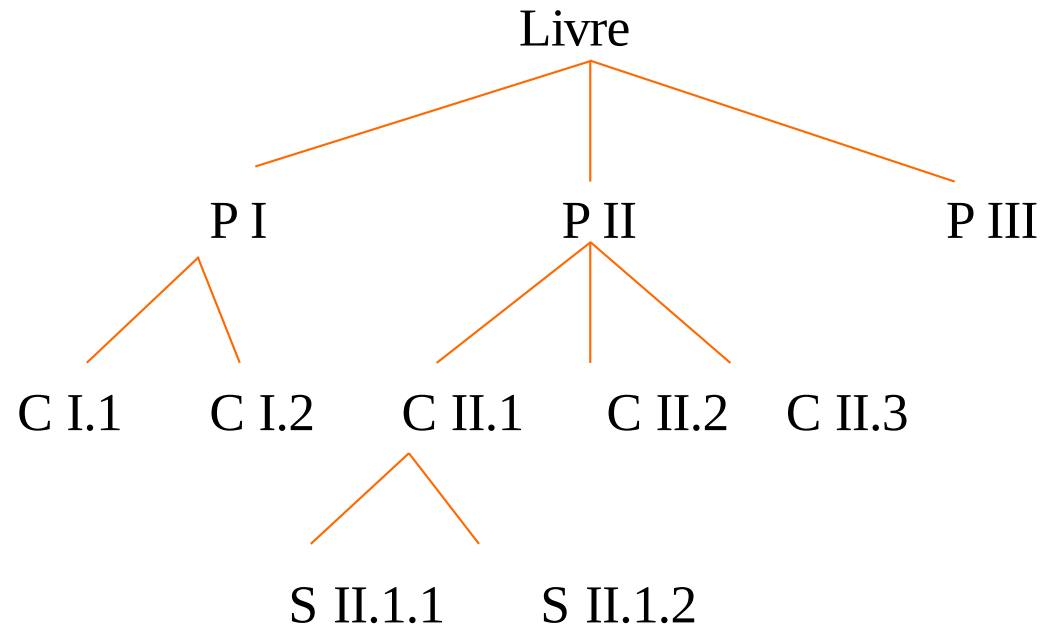
- ❑ Introduction
- ❑ Définitions et Terminologies
- ❑ Topologie
- ❑ Parcours

I. INTRODUCTION

- ❖ Les arbres sont des structures de données fondamentales en informatique, très utilisés dans tous les domaines, parce qu'ils sont bien adaptés à la représentation naturelle d'informations homogènes organisées, et d'une grande commodité et rapidité de manipulation.
- ❖ Leur usage est multiple, car il capte l'idée de hiérarchie; à titre d'exemples, nous pouvons citer:

I. INTRODUCTION

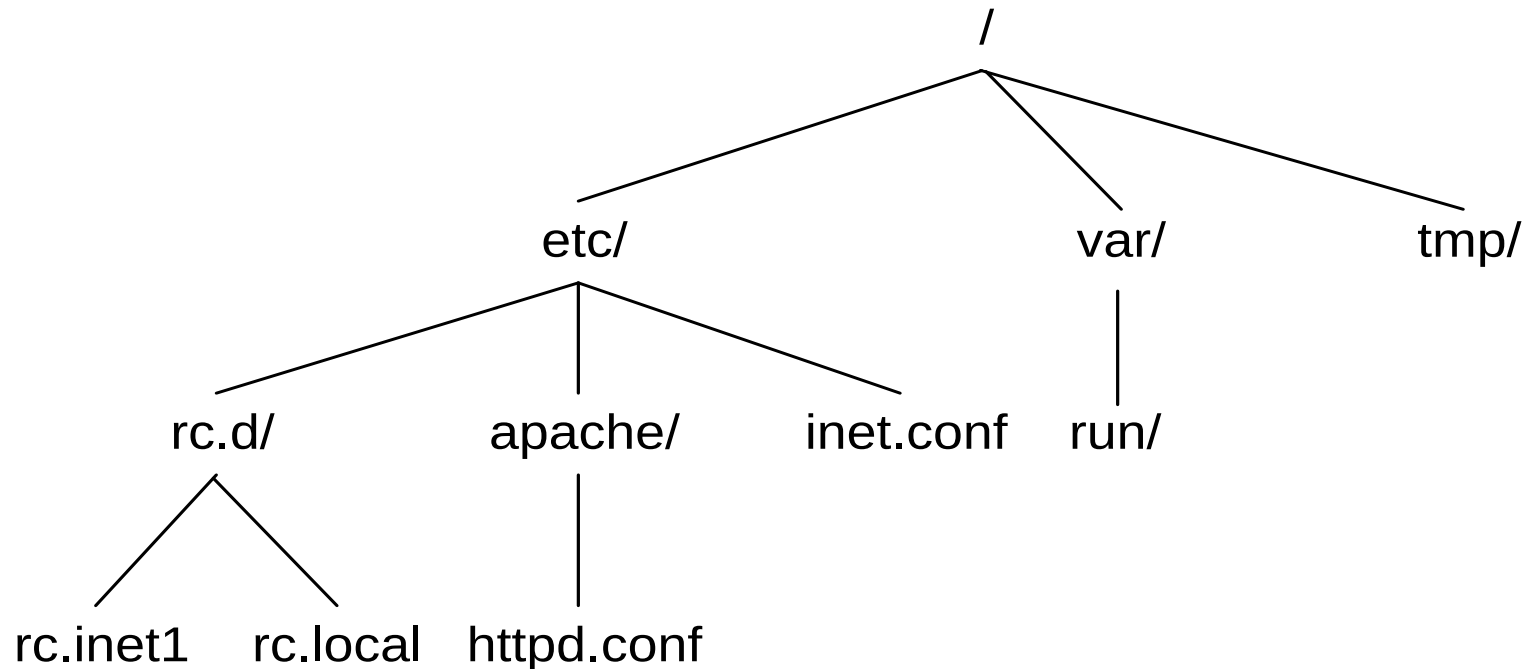
- ❖ Leur usage est multiple, car il capte l'idée de hiérarchie; à titre d'exemples, nous pouvons citer:
 - ❖ Découpage d'un livre en parties, chapitres, sections, paragraphes...,



I. INTRODUCTION

❖ Leur usage est multiple, car il capte l'idée de hiérarchie; à titre d'exemples, nous pouvons citer:

❖ Hiérarchies de fichiers,

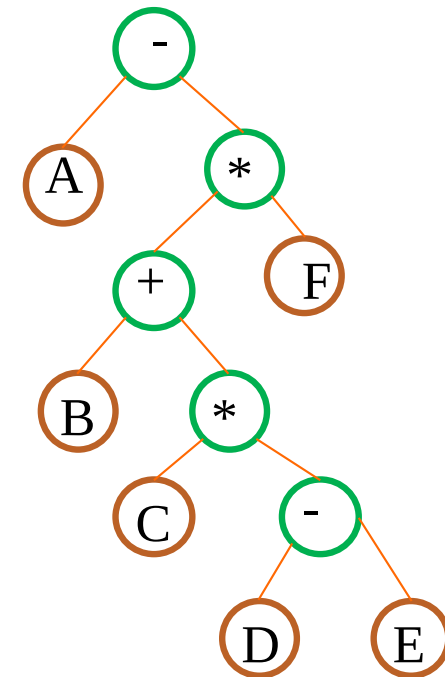


I. INTRODUCTION

❖ Leur usage est multiple, car il capte l'idée de hiérarchie; à titre d'exemples, nous pouvons citer:

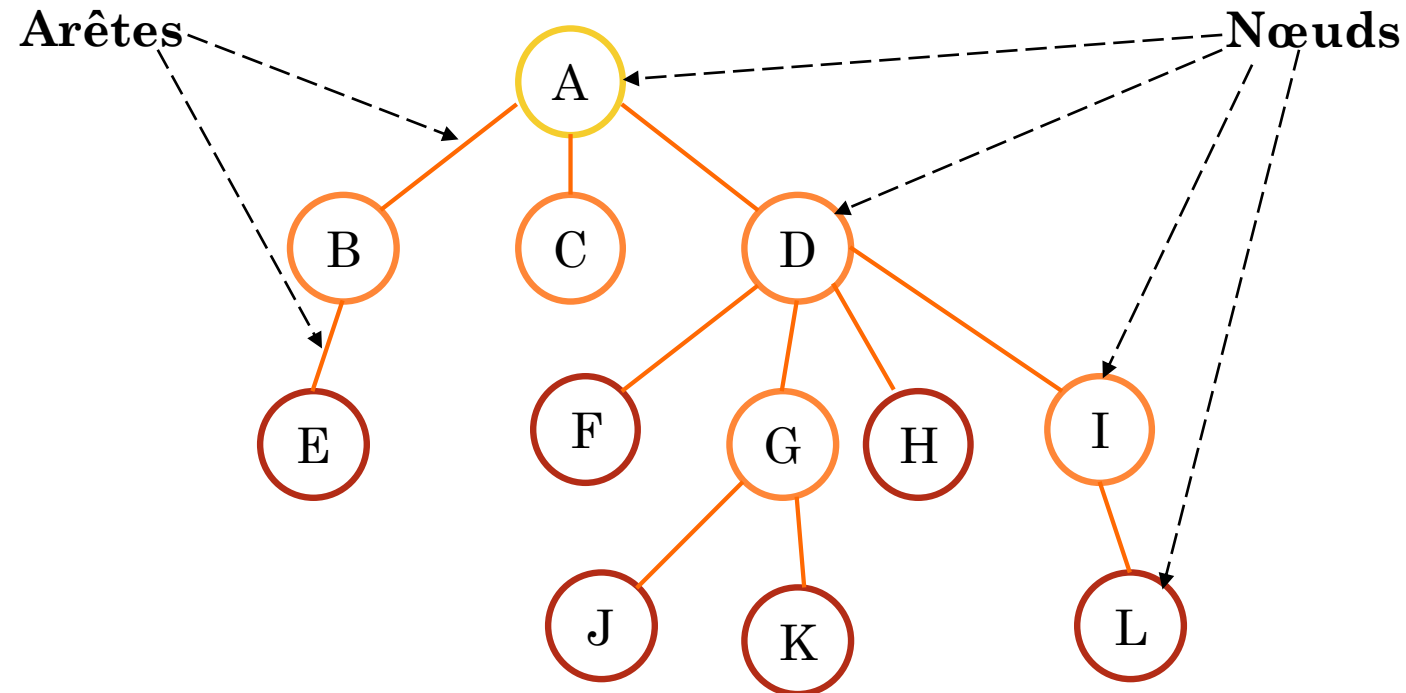
❖ Expressions Arithmétiques

▪ *L'expression $A - (B + C * (D - E)) * F$ se représente facilement par un arbre où apparaît clairement la priorité des opérations:*



I. DÉFINITIONS & TERMINOLOGIES

❖ Un arbre est une structure de données (souvent dynamique) représentant un ensemble de valeurs organisées hiérarchiquement (non linéaire). Chaque valeur est stockée dans un **nœud**. Les nœuds sont connectés entre eux par des **arêtes** qui représentent des relations parent/fils.

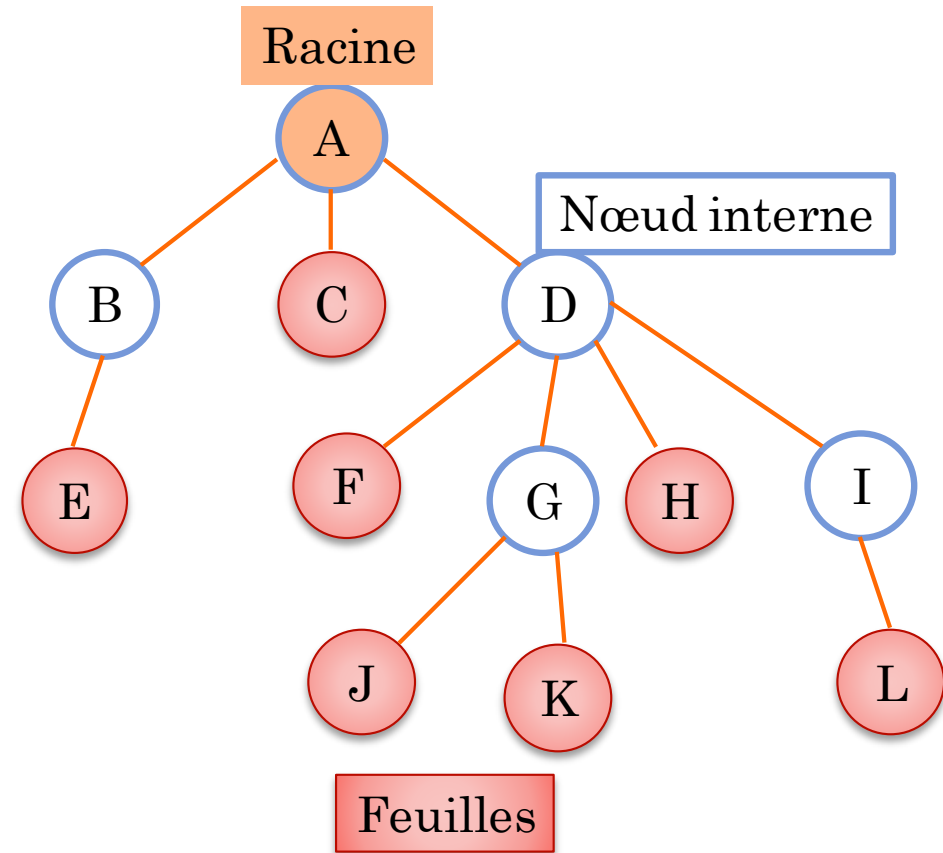


I. DÉFINITIONS & TERMINOLOGIES

❖ **Racine**: est le nœud qui n'a pas de **prédécesseur** (parent) et possède zéro ou plusieurs **fils**. La racine constitue la caractéristique d'un arbre.

❖ **Feuille** : est un nœud qui n'a pas de successeur (fils). Une feuille est aussi appelée un nœud externe.

❖ **Nœud interne** : est tout nœud qui admet au moins un successeur (fils).

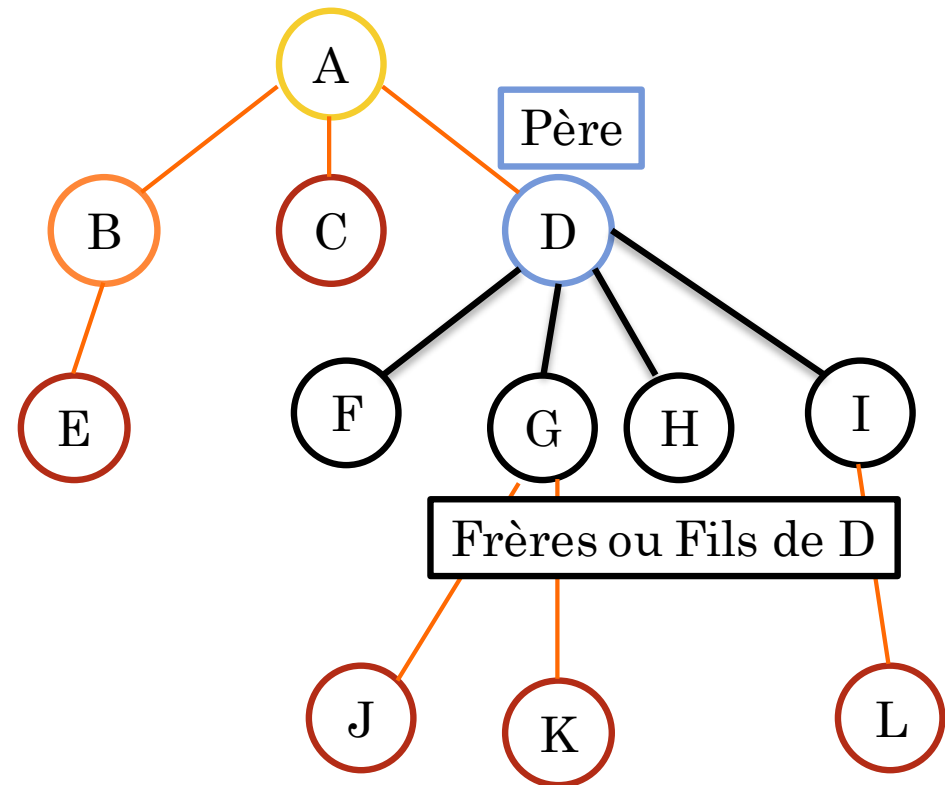


I. DÉFINITIONS & TERMINOLOGIES

❖ **Père** : est un nœud qui admet au moins un successeur (fils). Dans l'exemple, D est le père des nœuds F, G, H et I.

❖ **Fils d'un nœud** : sont ses successeurs. Dans l'exemple, F, G, H, et I sont les fils du nœud D.

❖ **Frères** : sont les successeurs ou les fils issus d'un même nœud (parent direct). Dans l'exemple, F, G, H et I sont des frères.



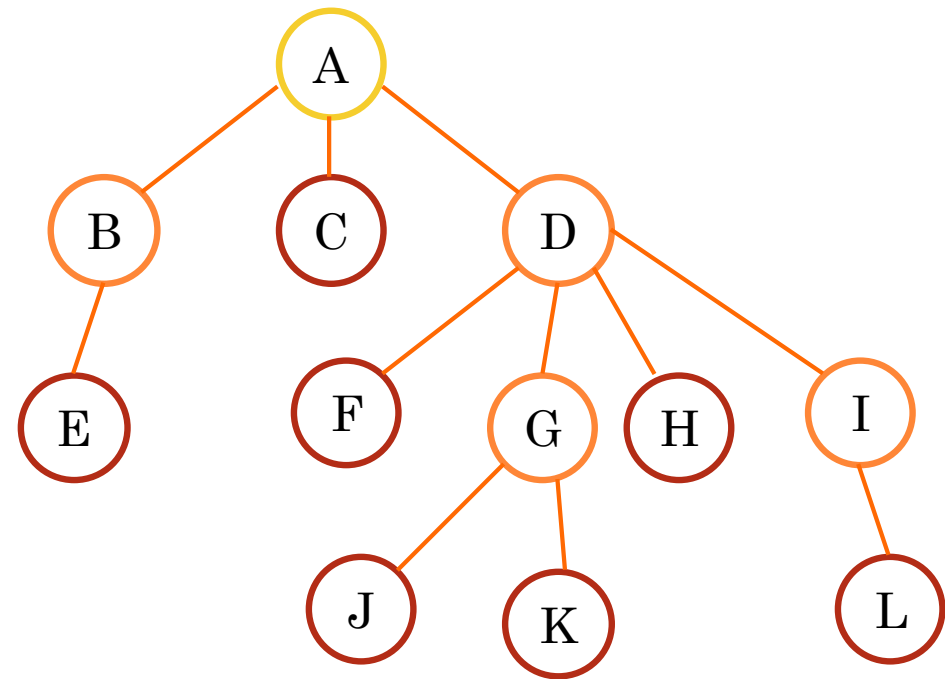
I. DÉFINITIONS & TERMINOLOGIES

❖ **Sous arbre** : est une portion de l'arbre.

Dans l'exemple, le nœud G avec ces deux fils J et K constituent un sous arbre.

❖ **Une branche** est une suite de nœuds connectés de père en fils (de la racine à une feuille).

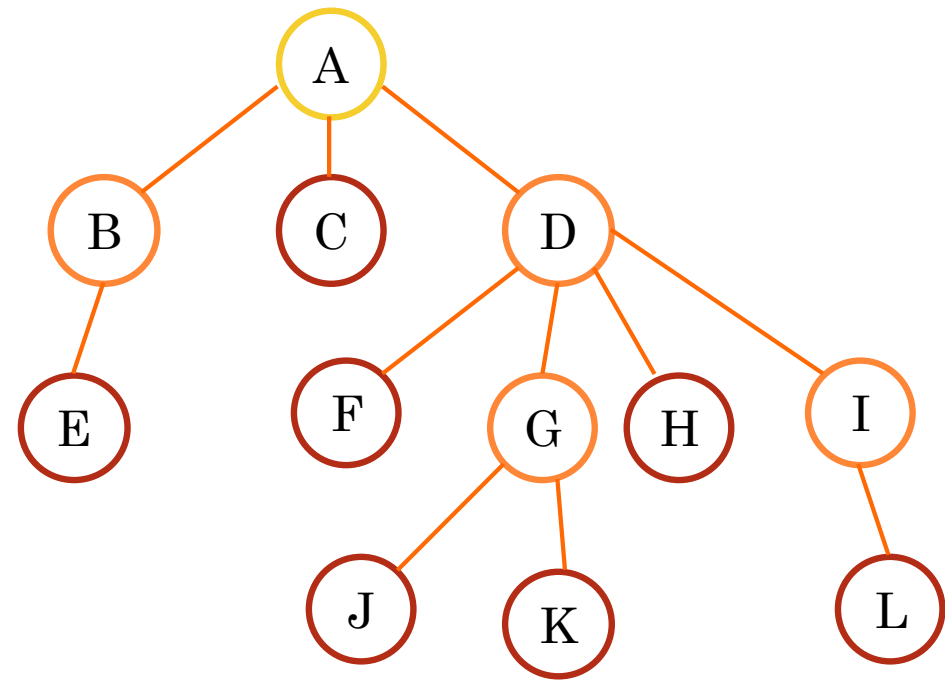
- A-B-E
- A-C
- A-D-F
- A-D-G-J
-



I. DÉFINITIONS & TERMINOLOGIES

❖ **Descendants d'un nœud** : sont tous les nœuds du sous arbre de racine nœud. Dans l'exemple, les descendants de D sont F, G, H, I, J, K et L.

❖ **Ascendants d'un nœud** : sont tous les nœuds se trouvant sur la branche de la racine vers ce nœud. Dans l'exemple, les ascendants de J sont G, D et A. Les ascendants de E sont B et A.



I. DÉFINITIONS & TERMINOLOGIES

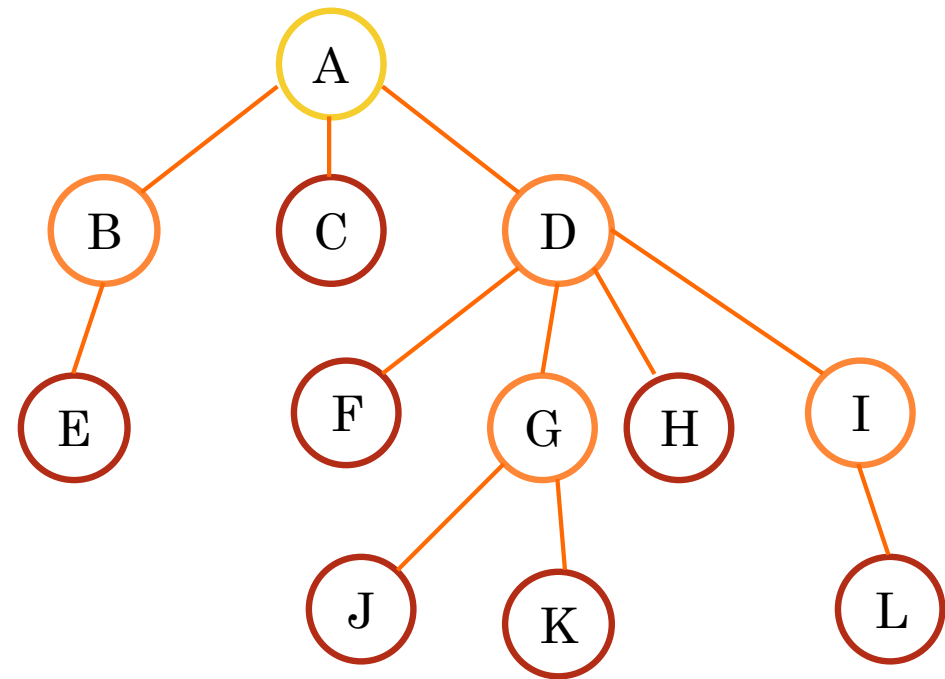
❖ **Taille d'un arbre:** est le nombre de nœuds qu'il possède.

- Taille de l'arbre ci contre = 12
- Un arbre vide est de taille égale à 0.

❖ **Degré d'un nœud :** est le nombre de ses fils. Dans l'exemple, le degré de B est 1, le degré de D est 4.

❖ **Degré d'un arbre :** est le degré maximum de ses nœuds.

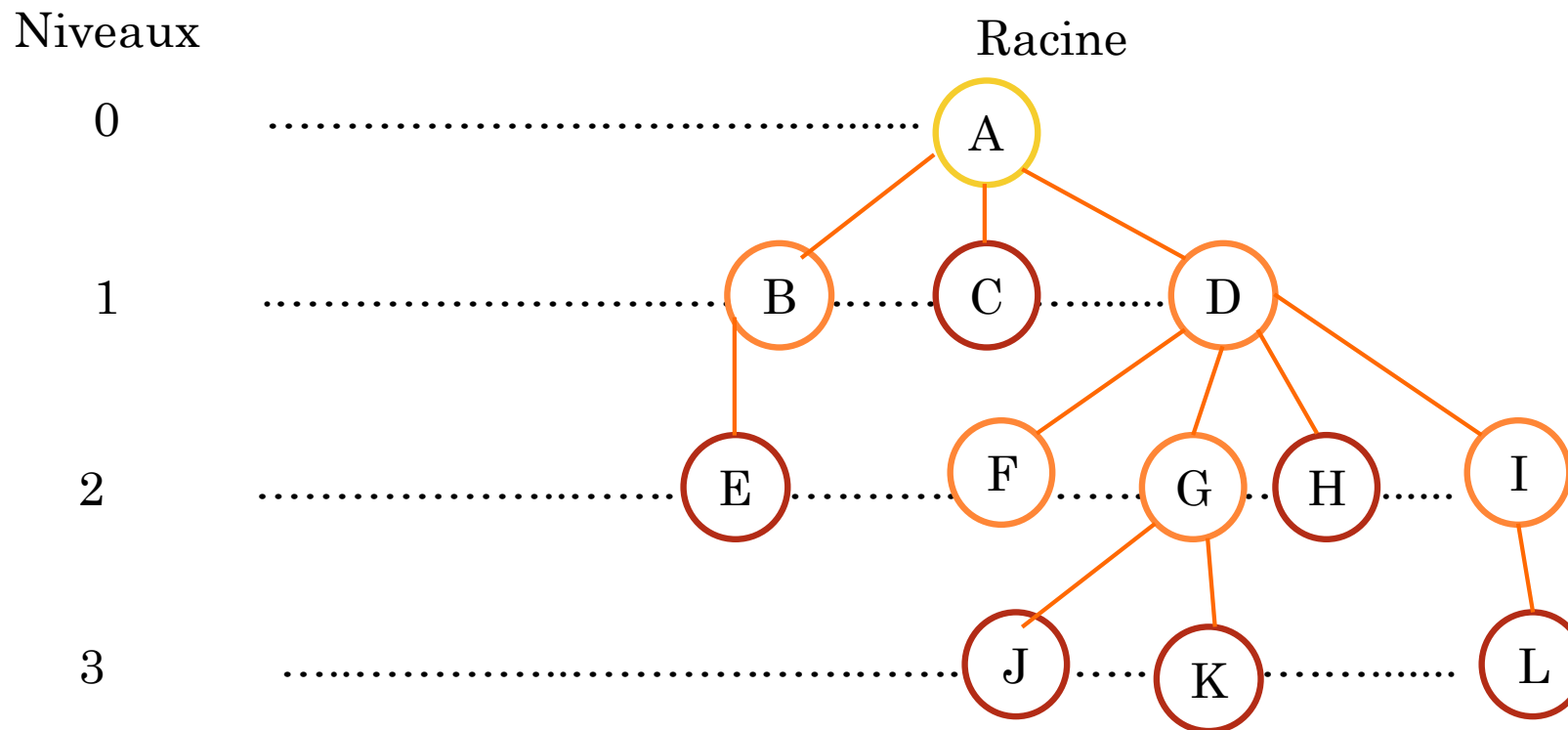
- Degré de l'arbre ci contre = 4



I. DÉFINITIONS & TERMINOLOGIES

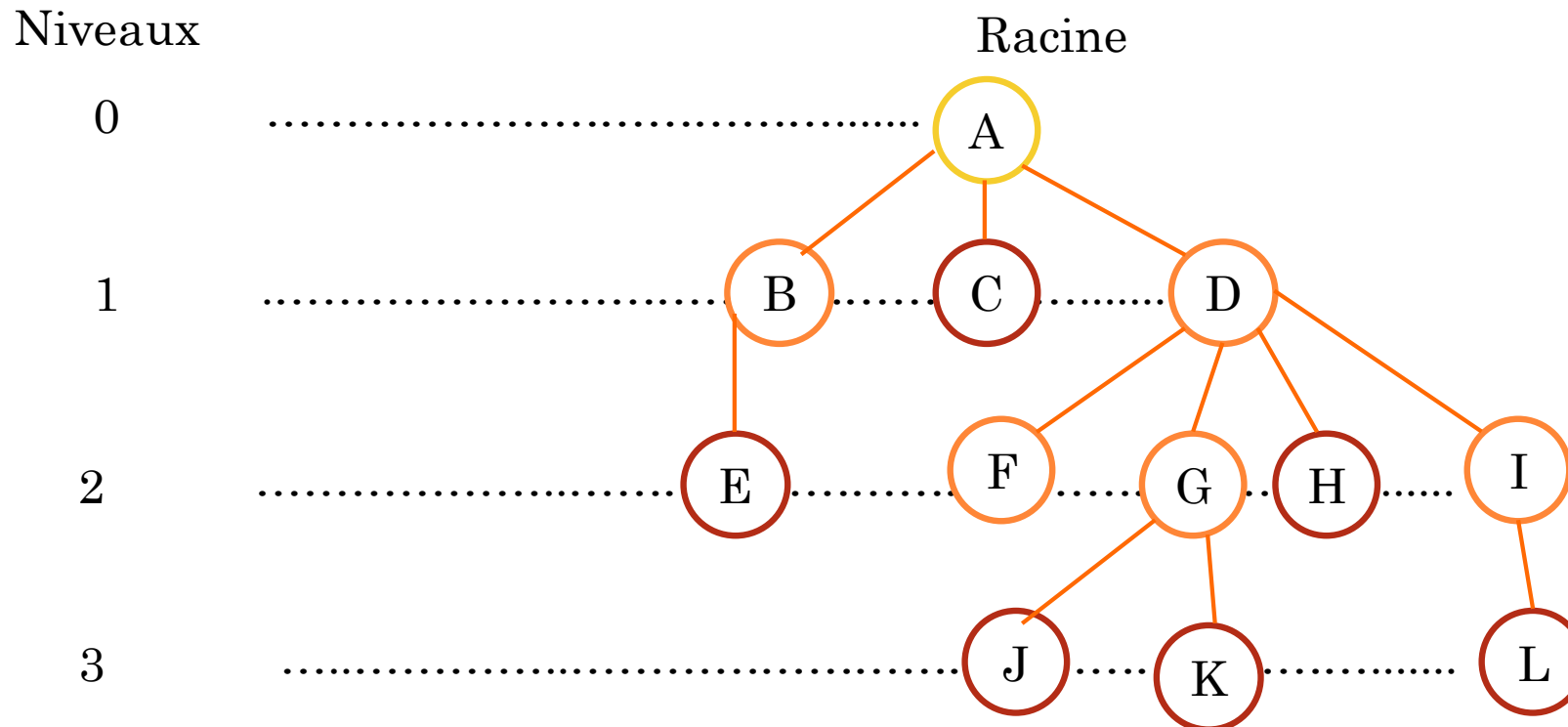
❖ **Le niveau d'un nœud:** est la distance qui le sépare de la racine:

- Le niveau de la racine = 0
- Le niveau de chaque nœud est égale au niveau de son père plus 1
- Le niveau du nœud contenant 'G' est égal à 2.



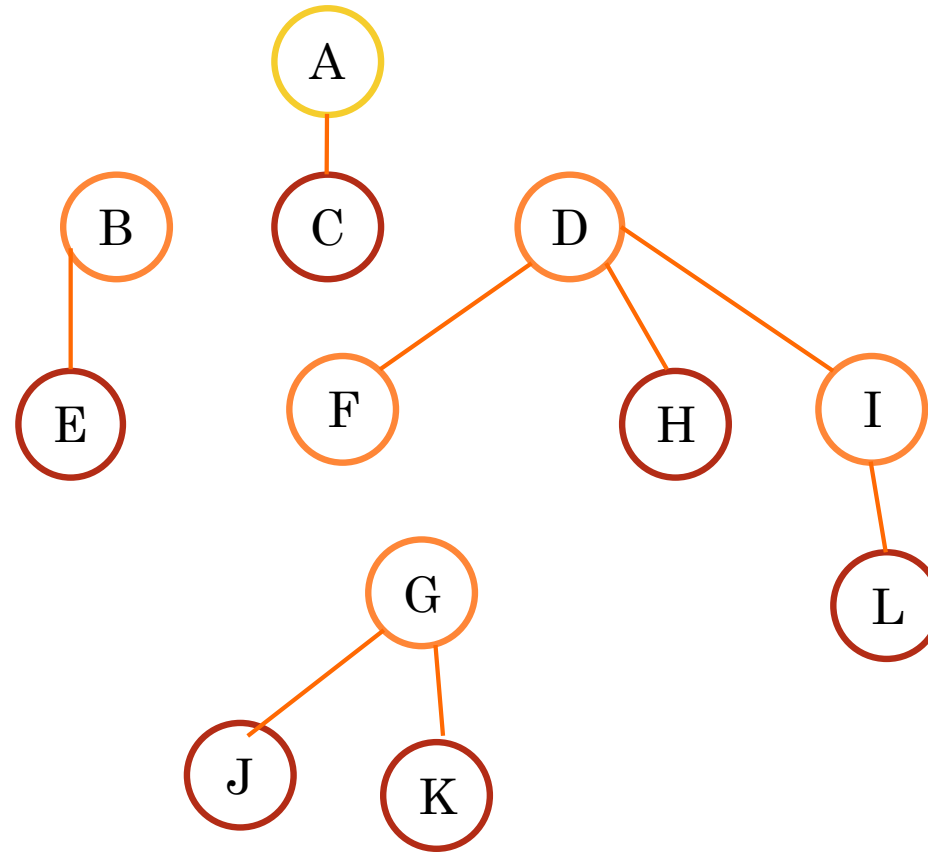
I. DÉFINITIONS & TERMINOLOGIES

❖ **La profondeur (hauteur) d'un arbre** : est le plus grand niveau, c.à.d la distance entre la racine et la feuille la plus lointaine. Dans l'exemple ci-dessus, la profondeur de l'arbre est égal à 3



I. DÉFINITIONS & TERMINOLOGIES

❖ **Forêt** : est un ensemble d'arbres.



I. TYPOLOGIE

❖ Les arbres sont classifiés selon:

- **leur degré $m \geq 2$:** On trouve les arbres binaires ($m = 2$), ternaires ($m=3$), quaternaires ($m = 4$),, m-aire ($m > 2$).
- **la priorité d'ordre:** entre le nœud père et ses fils, où :
 - ✓ le nœud père possède au moins une clé, les valeurs de sous arbre gauche (droit resp.) de la clé sont strictement inférieures (supérieures ou égales resp.) à la valeur de la clé. On parle ici des arbres binaires de recherche (où chaque nœud possède une clé) et des arbres m-aire de recherche (où chaque nœud possède $(m-1)$ clé),
 - ✓ la valeur du nœud père est supérieure (inférieure resp.) à la valeur de ses fils. On parle ici du $TAS_{\max}$ ($TAS_{\min}$)

I. TYPOLOGIE

❖ Les arbres sont classifiés selon:

❖ **La propriété d'équilibrage:** où les feuilles se situent aux derniers niveaux. On trouve les arbres binaires de recherche équilibrés (e.g: AVL, TAS, rouge et noir, 2-3-4,) et les arbre m-aire de recherche équilibré (B-arbres, B-arbre*, B-Arbre+,))

I. PARCOURS

- ❖ Le parcours d'un arbre consiste à passer par tous ses nœuds.
- ❖ Les parcours permettent d'effectuer tout un ensemble de traitement sur les arbres.
- ❖ On distingue deux types de parcours :
 - Des parcours en **profondeur** (depth-first) explorent l'arbre branche par branche. Parmi lesquels: le Préordre, l'Inordre et le Postordre.
 - Des parcours en **largeur** (breadth-first) explorent l'arbre niveau par niveau

I. PARCOURS

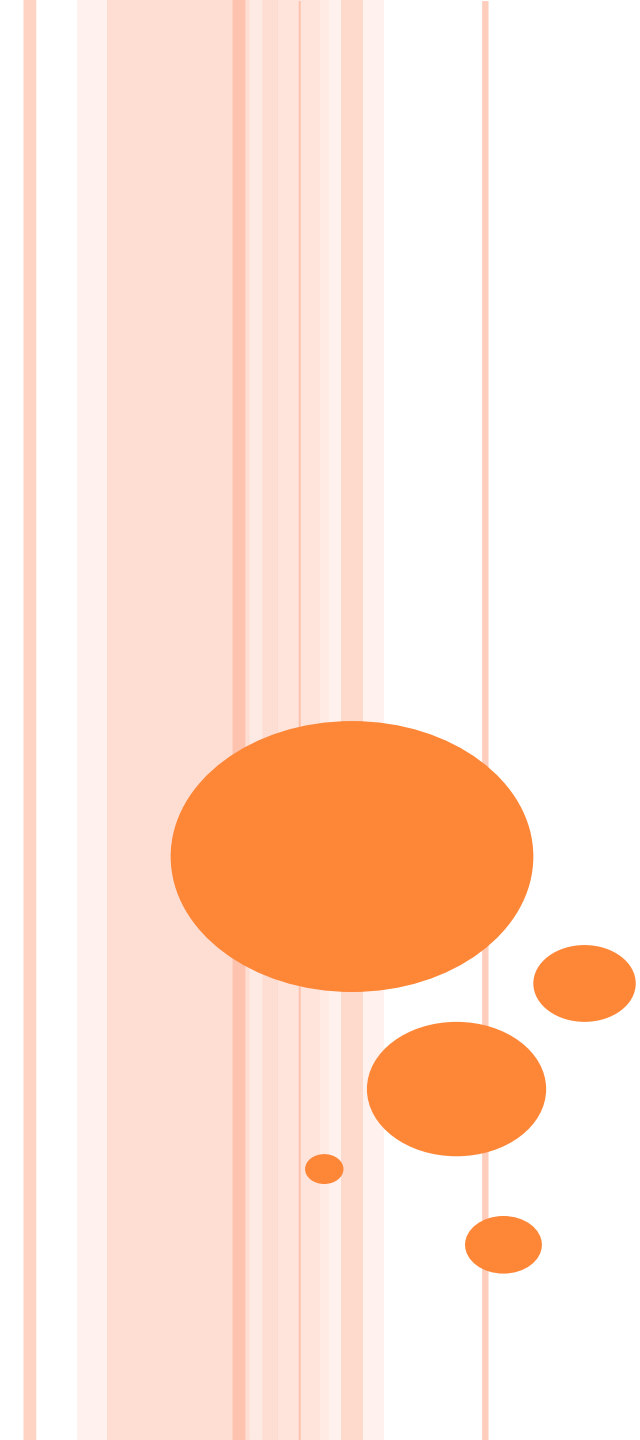
❖ Dans un parcours en profondeur, on descend le plus profondément possible dans l'arbre puis, une fois qu'une feuille a été atteinte, on remonte pour explorer les autres branches en commençant par la branche « la plus basse » parmi celles non encore parcourues.

❖ Le parcours en profondeur peut se faire en :

❖ **Préordre (Préfixe)** : où on affiche la racine avant ses fils,

❖ **Postordre(Postfixe)** : où on affiche les fils avant leur racine.

❖ **Inordre (Infixe)** : où, dans le cas d'un arbre binaire, on affiche le fils gauche avant sa racine et son frère droit,



PARTIE II:

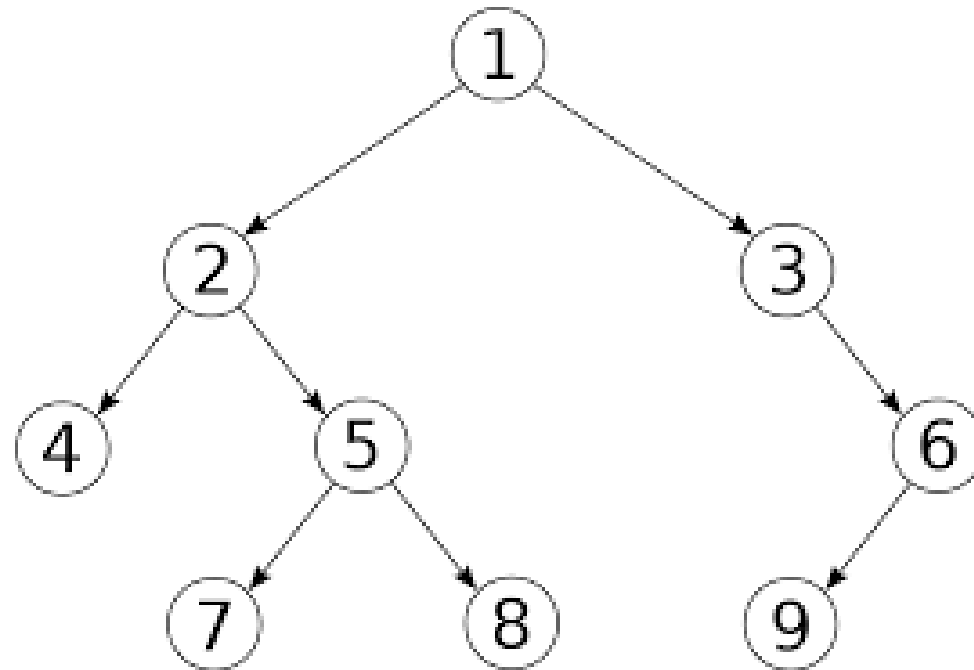
ARBRES BINAIRES DE RECHERCHE (ABR)

PLAN DE LA PARTIE II

- ❑ Définitions
- ❑ Parcours
- ❑ Opérations de Base: Recherche, Insertion et Suppression

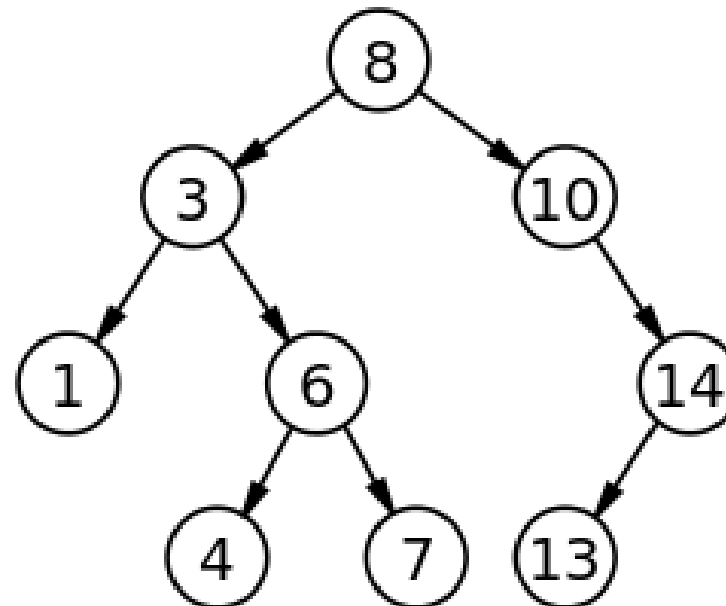
II. DÉFINITION

❖ Un **Arbre Binaire** est un arbre de degré 2, c'est-à-dire que chaque nœuds a au plus deux fils. Ainsi, le premier fils d'un nœud est appelé Fils-Gauche (FG) et le deuxième fils est appelé Fils-Droit (FD).



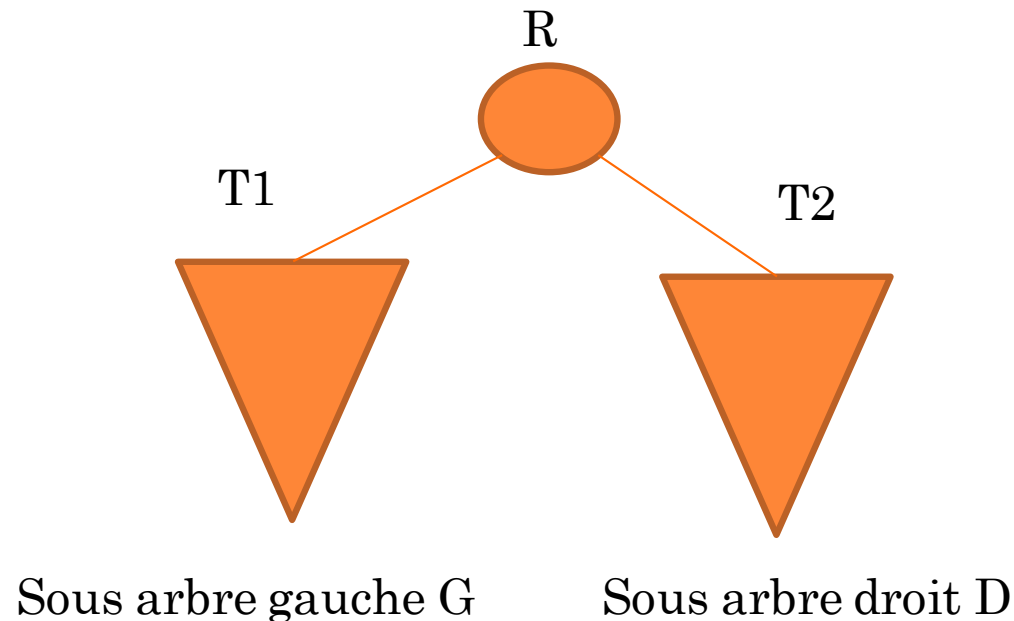
II. DÉFINITION

- ❖ Un **Arbre Binaire de Recherche (ABR)** est un arbre binaire ordonné tel que pour tout nœud « i »:
 - Toutes les valeurs du sous arbre gauche de « i » sont strictement inférieures à la valeur de « i », et
 - Toutes les valeurs du sous-arbre droit de « i » sont supérieures ou égales à la valeur de « i ».



II. PARCOURS PREORDRE

❖ Le parcours préordre d'un arbre binaire R consiste à visiter le nœud racine (R) ensuite parcourir récursivement en préordre les sous arbres T1 (sous arbre gauche) puis T2 (sous arbre droit) ce qui donne : **[R , T1 , T2 ou RGD]**

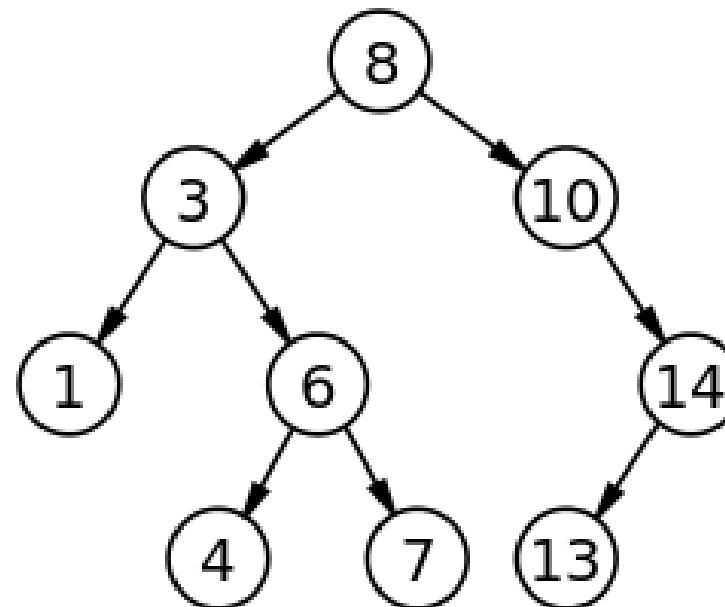


II. PARCOURS PREORDRE

❖ Le parcours préordre d'un arbre binaire R consiste à visiter le nœud racine (R) ensuite parcourir récursivement en préordre les sous arbres T1 (sous arbre gauche) puis T2 (sous arbre droit) ce qui donne : **[R , T1 , T2 ou RGD]**

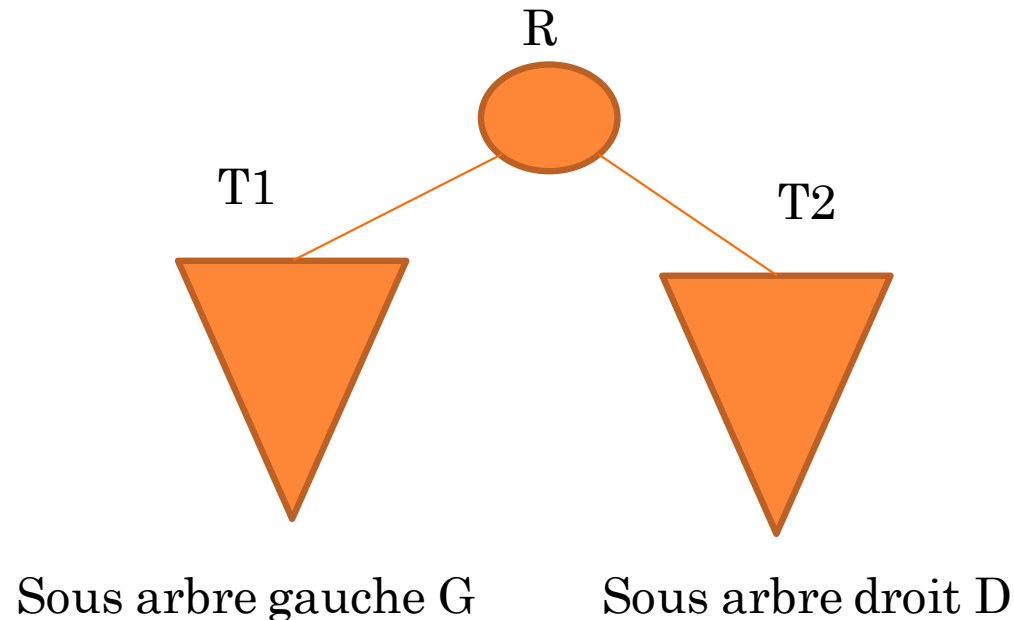
Résultat de parcours:

8, 3, 1, 6, 4, 7, 10, 14, 13



II. PARCOURS INORDRE

❖ Le parcours inordre d'un arbre binaire R consiste d'abord à parcourir récursivement en inordre le sous arbre gauche T1, puis visiter le nœud racine (R) ensuite parcourir récursivement en inordre le sous arbre droit T2 ce qui donne [T1 , R , T2 ou GRD]



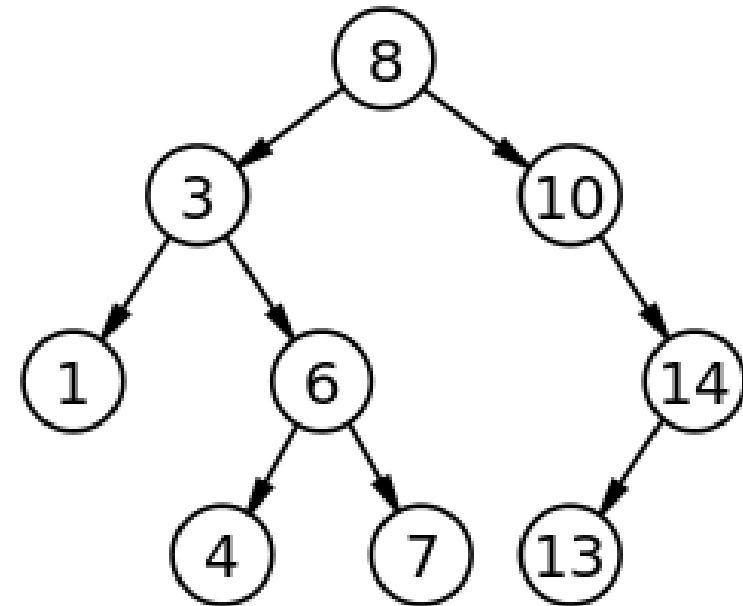
II. PARCOURS INORDRE

❖ Le parcours inordre d'un arbre binaire R consiste d'abord à parcourir récursivement en inordre le sous arbre gauche T1, puis visiter le nœud racine (R) ensuite parcourir récursivement en inordre le sous arbre droit T2 ce qui donne [T1 , R , T2 ou GRD]

Résultat de parcours:

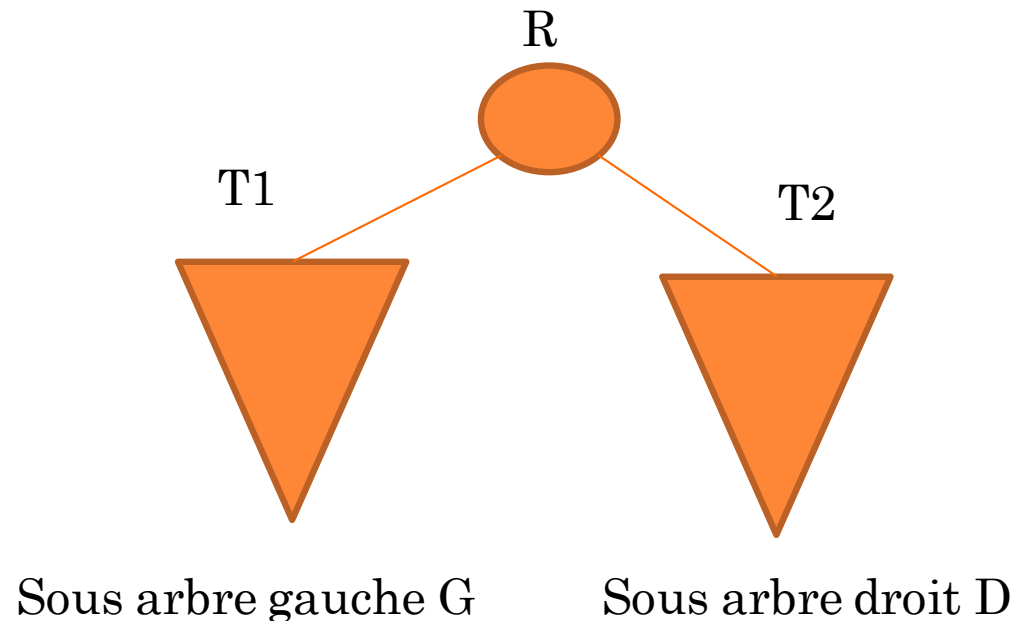
1, 3, 4, 6, 7, 8, 10, 13, 14

❖ Propriété: Le parcours inordre d'un ABR visite les nœuds par ordre croissant des clés.



II. PARCOURS POSTORDRE

❖ Le parcours postordre d'un arbre binaire R consiste d'abord à parcourir récursivement en postordre les sous arbres T1 puis T2 ensuite visiter le nœud racine (R) ce qui donne [T1 , T2 , R ou GDR]

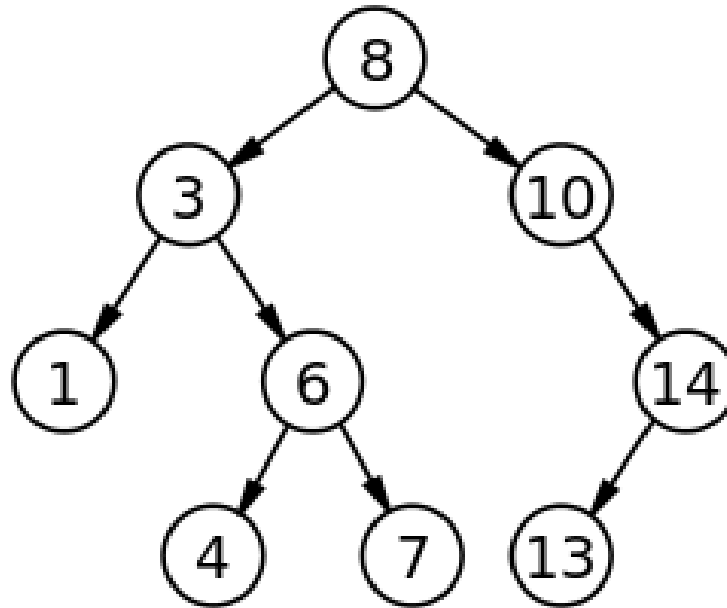


II. PARCOURS POSTORDRE

❖ Le parcours postordre d'un arbre binaire R consiste d'abord à parcourir récursivement en postordre les sous arbres T1 puis T2 ensuite visiter le nœud racine (R) ce qui donne [T1 , T2 , R ou GDR]

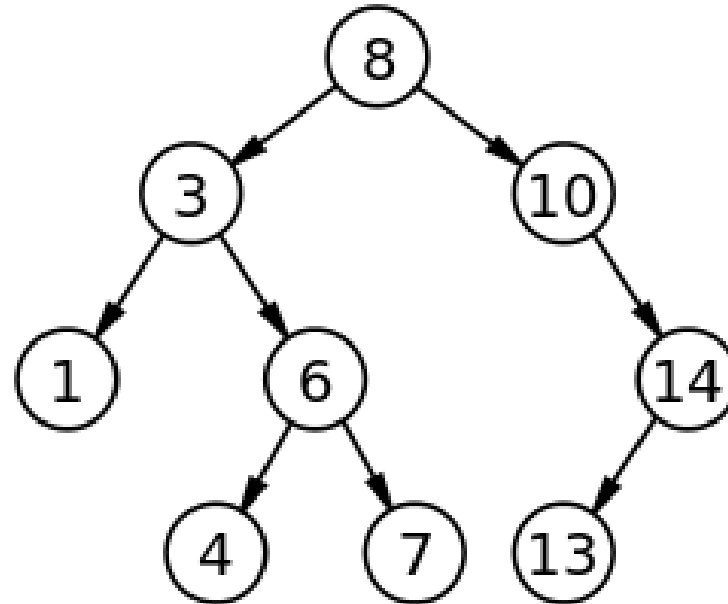
Résultat de parcours:

1, 4, 7, 6, 3, 13, 14, 10, 8



II. PARCOURS EN LARGEUR

- ❖ Dans le parcours par niveau, tous les nœuds d'un même niveau sont traités avant de descendre au niveau suivant



Résultat de parcours: 8, 3, 10, 1, 6, 14, 4, 7, 13

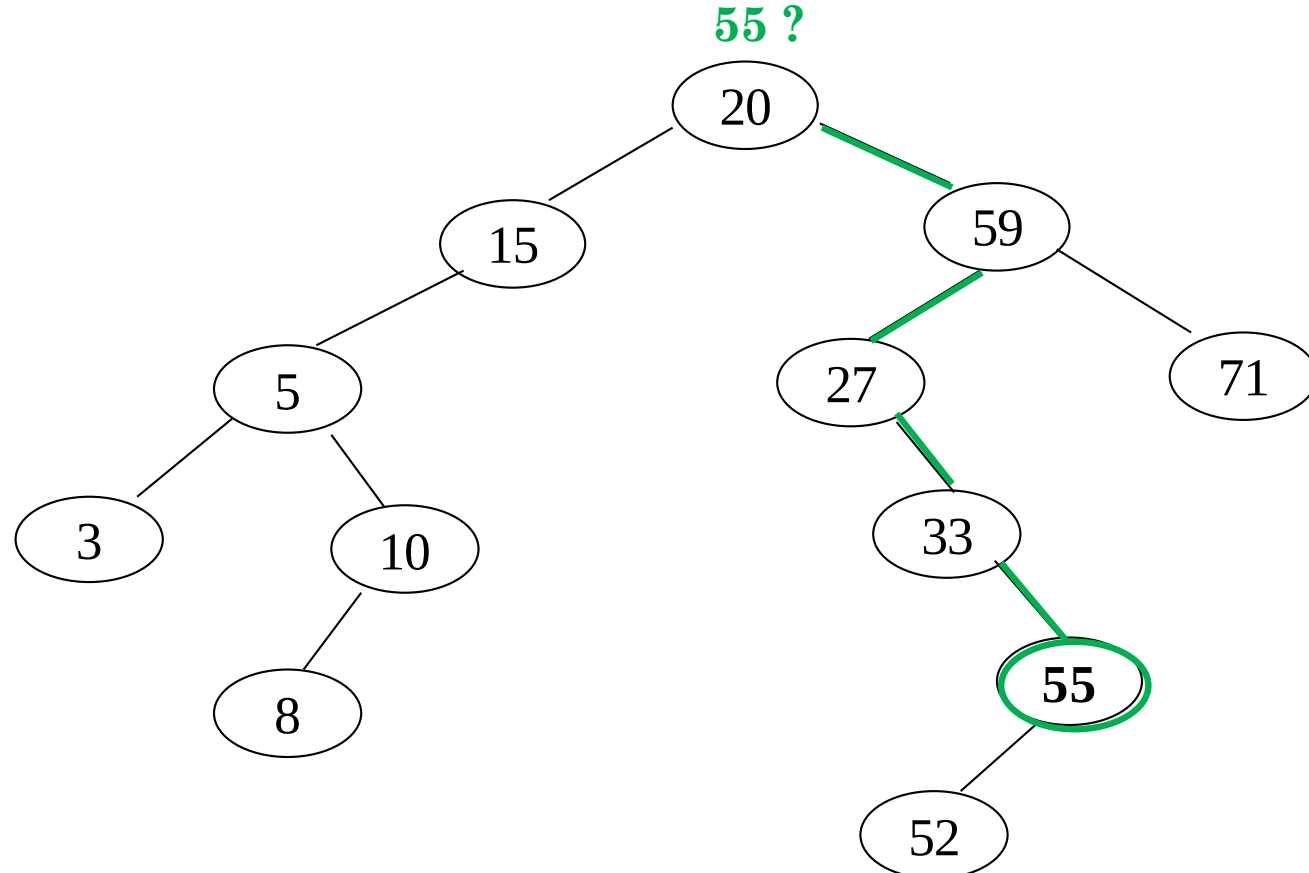
II. OPÉRATION DE RECHERCHE

- ❖ La recherche d'un nœud ayant une clé particulière « x » se fait de manière récursive:
 - Si l'arbre est vide: échec
 - ✓ Recherche négative: la clé n'appartient pas à l'arbre, l'algorithme se termine.
 - Sinon: comparer la clé « x » avec la clé de la racine.
 - S'il y a égalité alors la clé « x » est trouvée.
 - ✓ Recherche positive: l'algorithme se termine.
 - Si la clé « x » est strictement inférieure, alors la recherche est poursuivie dans le sous arbre gauche.
 - Si la clé « x » est strictement supérieure, alors la recherche est poursuivie dans le sous arbre droit.

II. OPÉRATION DE RECHERCHE

❖ La recherche est **dichotomique**, à chaque étape, un sous arbre est éliminé:

- Rechercher (55)
- Rechercher (FD(20))
- Rechercher (FG(59))
- Rechercher (FD(27))
- Rechercher (FD (33))
- Élément trouvé



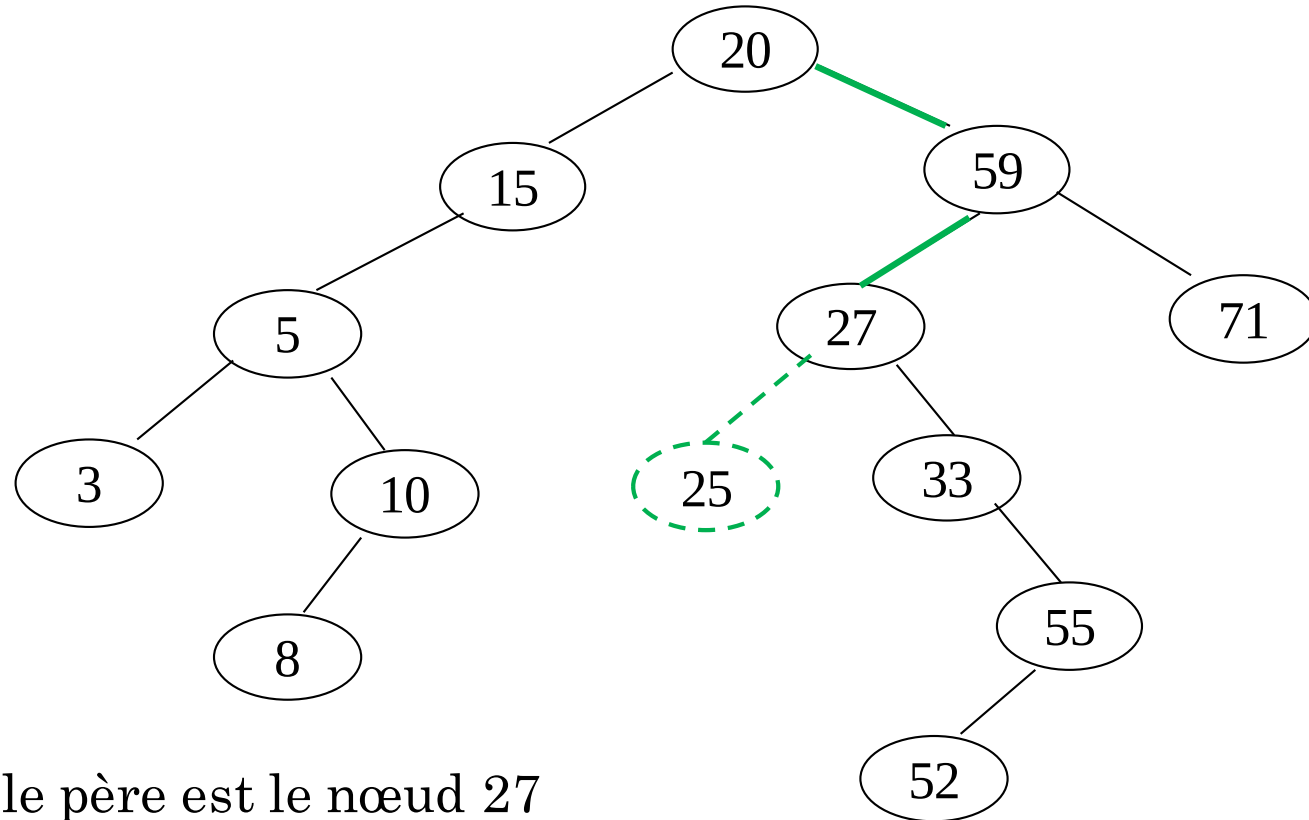
II. OPÉRATION D'INSERTION

❖ L'insertion d'une clé « x » se fait toujours au niveau d'une **feuille**. Cette insertion dans un ABR doit maintenir la propriété d'ordre des arbres de recherche, ainsi:

1. Rechercher la position d'insertion
2. Raccorder le nouveau nœud à son parent

II. OPÉRATION D'INSERTION

+ 25



- RechercherPosition (25)
- Rechercher (FD(20))
- Rechercher (FG(59))
- Position trouvé pour l'insertion, le père est le nœud 27
- Insérer 25 au niveau de la feuille dont le père est 27

II. OPÉRATION DE SUPPRESSION

❖ La suppression d'une clé « x » passe par deux étapes:

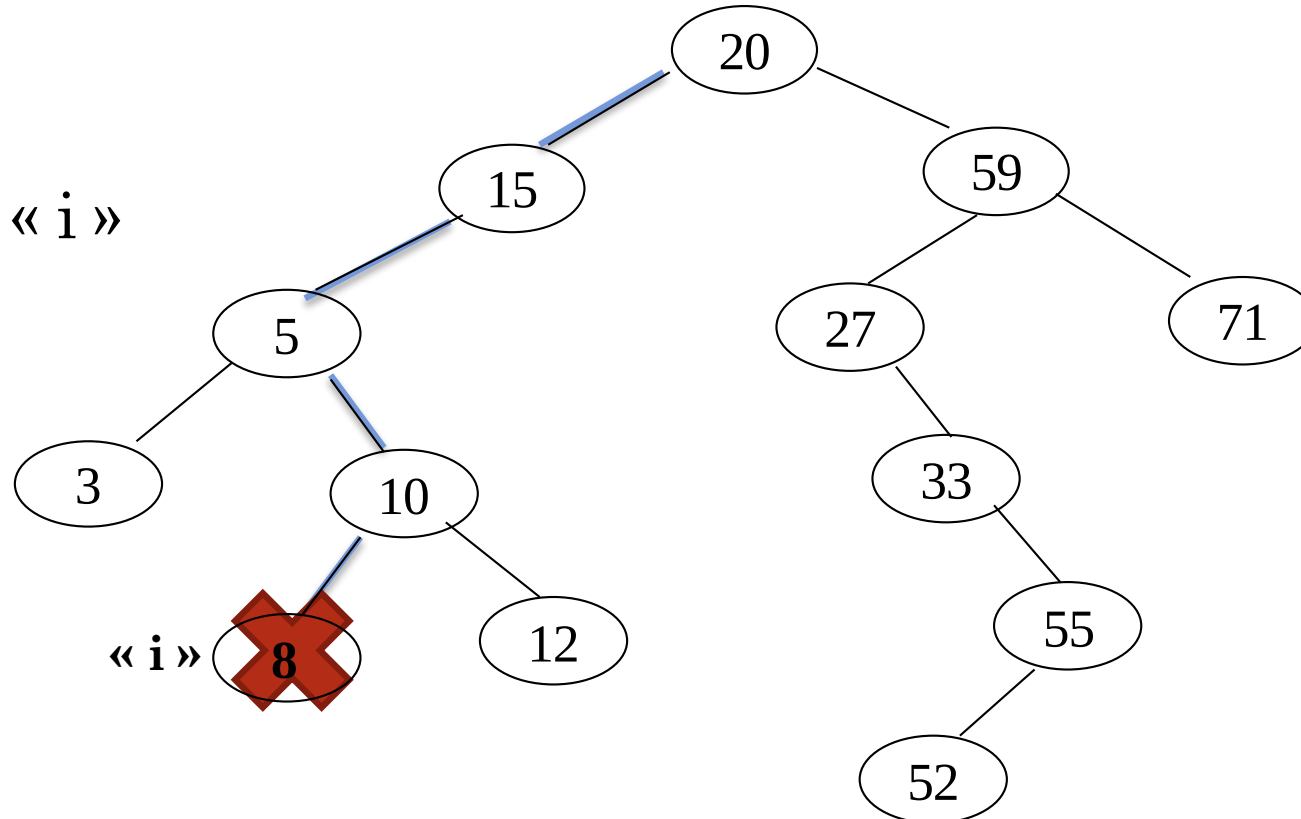
1. Rechercher d'abord le nœud qui contient la clé « x »; soit « i » ce nœud
2. Supprimer le nœud « i » tout en maintenant la propriété de l'ordre de l'ABR. Ainsi, on peut se trouver dans une des situations suivantes :

II. OPÉRATION DE SUPPRESSION

❖ Cas 1: Suppression d'une feuille

- Il suffit de l'enlever de l'arbre vu qu'elle n'a pas de fils.
- Exemple: supprimer le nœud i qui contient la valeur 8

1. Rechercher(8)
2. Libérer le nœud « i »

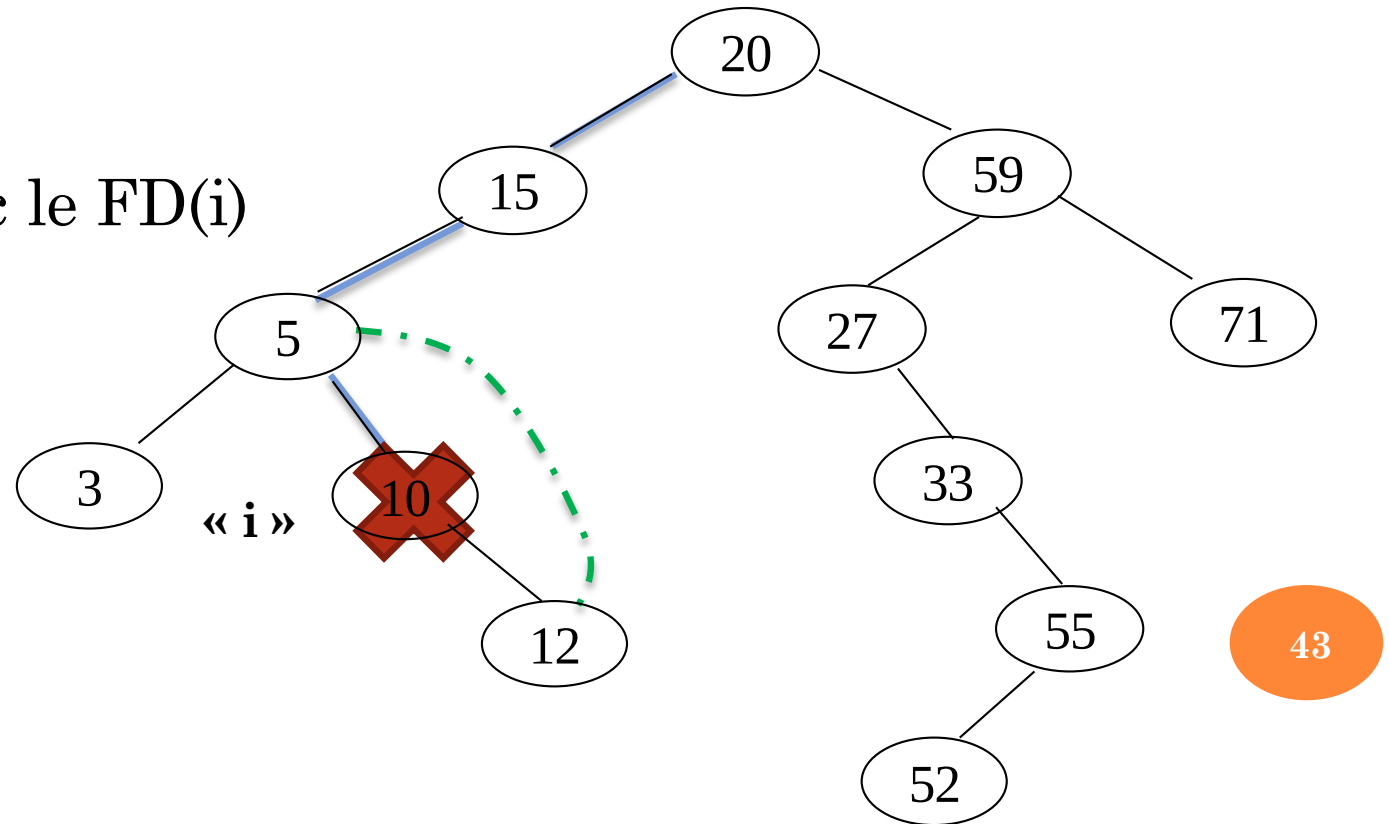


II. OPÉRATION DE SUPPRESSION

❖ Cas 2: Suppression d'un nœud avec un fils

- Il suffit de l'enlever de l'arbre en liant son père avec son fils.
- Exemple: supprimer le nœud « i » qui contient la valeur 10

1. Rechercher(10)
2. Chainer le père de i avec le FD(i)
3. Libérer le nœud « i »



II. OPÉRATION DE SUPPRESSION

❖ Cas 2: Suppression d'un nœud avec un fils

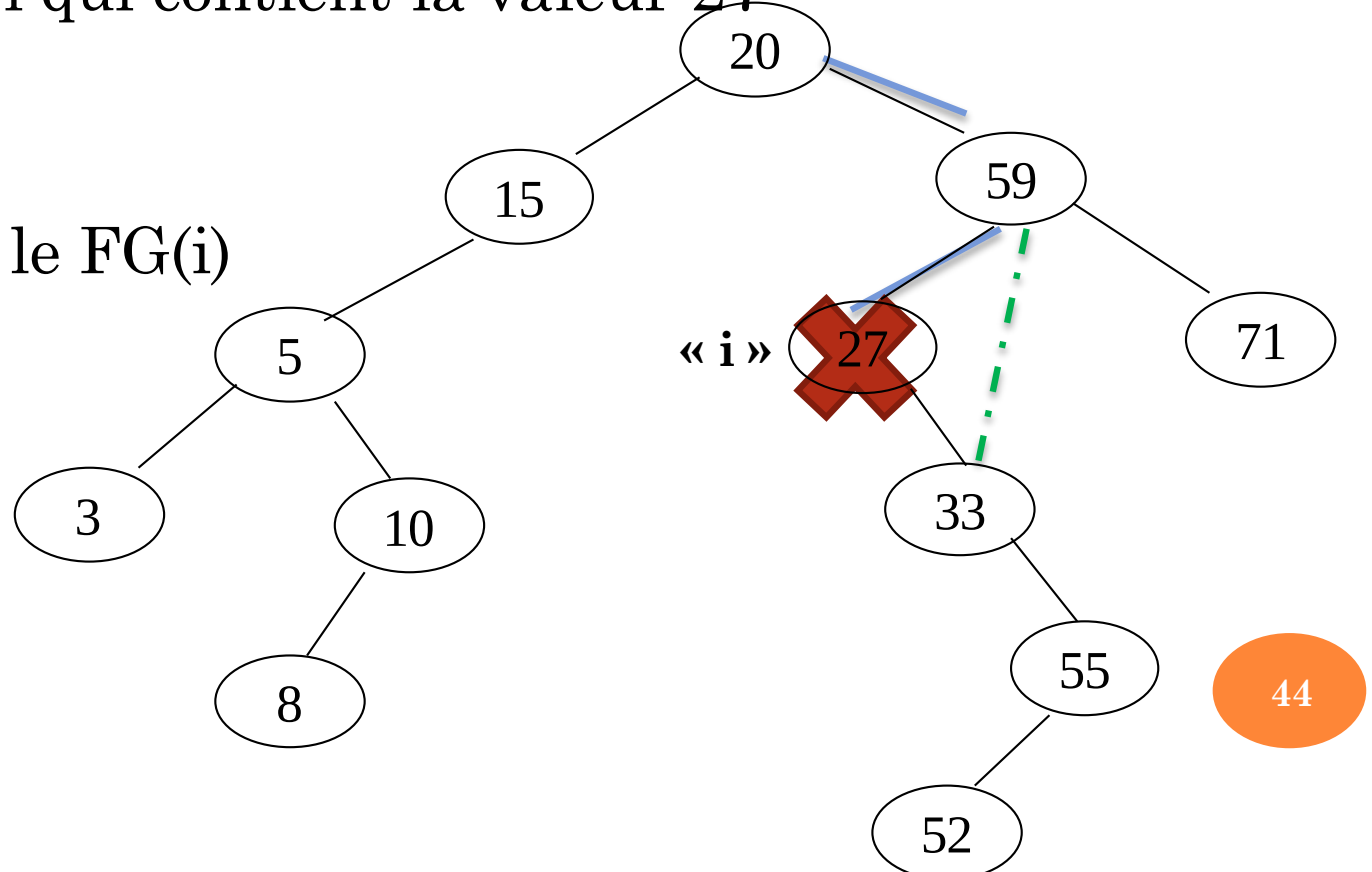
➤ Il suffit de l'enlever de l'arbre en liant son père avec son fils.

➤ Exemple: supprimer le nœud i qui contient la valeur 27

1. Rechercher(27)

2. Chainer le père de i avec le FG(i)

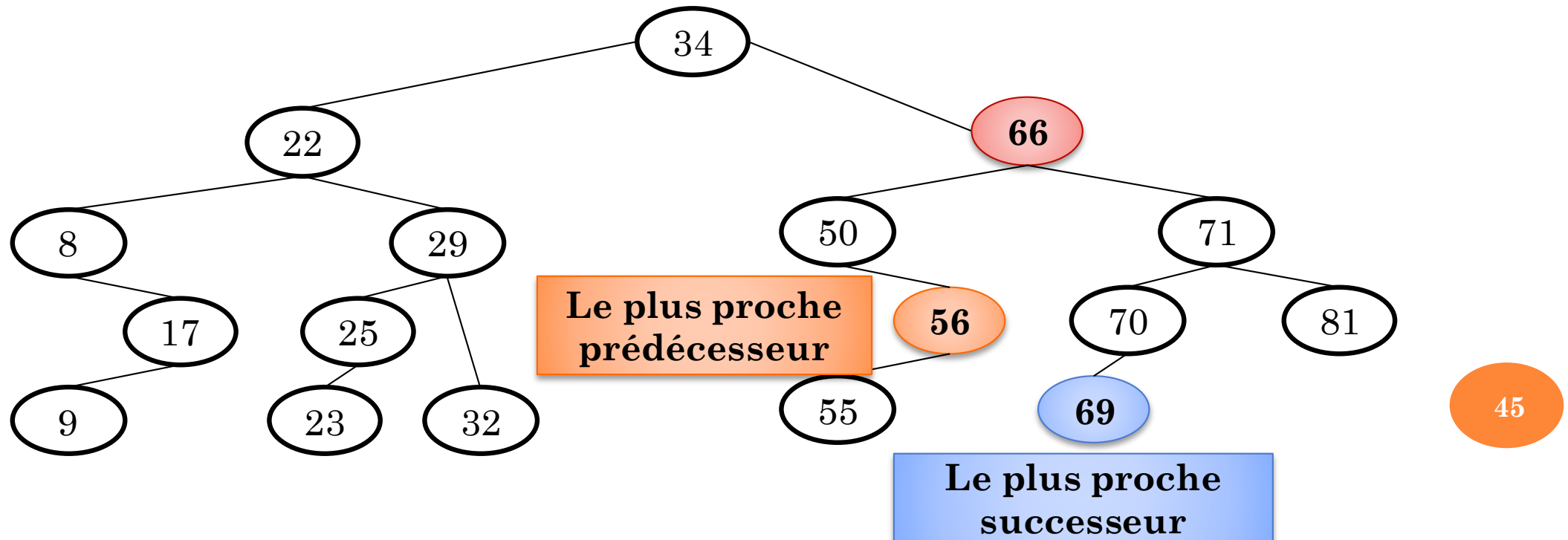
3. Libérer le nœud « i »



II. OPÉRATION DE SUPPRESSION

❖ Cas 3: Suppression d'un nœud avec deux fils

- **Etape 1:** On échange le nœud à supprimer avec son successeur le plus proche (le nœud le plus à gauche du sous-arbre droit) ou son plus proche prédécesseur (le nœud le plus à droite du sous-arbre gauche). Cela permet de garder la propriété d'ordre d'ABR.

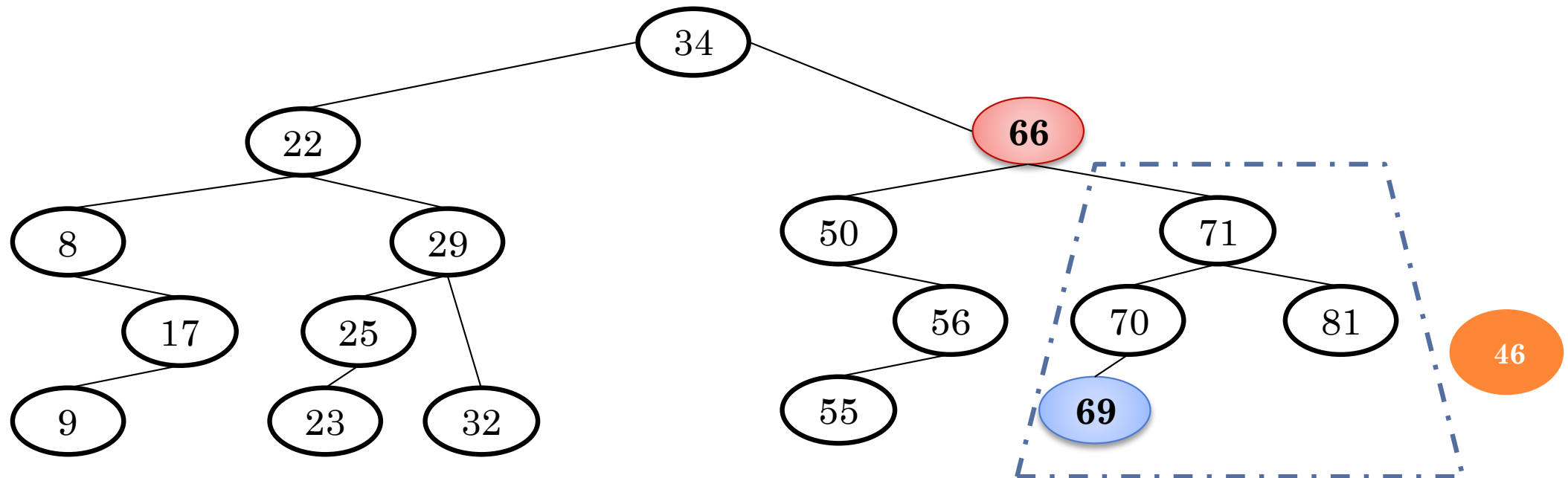


II. OPÉRATION DE SUPPRESSION

❖ Cas 3: Suppression d'un nœud avec deux fils

➤ **Etape 1 → Cas A:** On échange le nœud à supprimer avec son successeur le plus proche (le nœud le plus à gauche ou le plus petit du sous arbre droit)

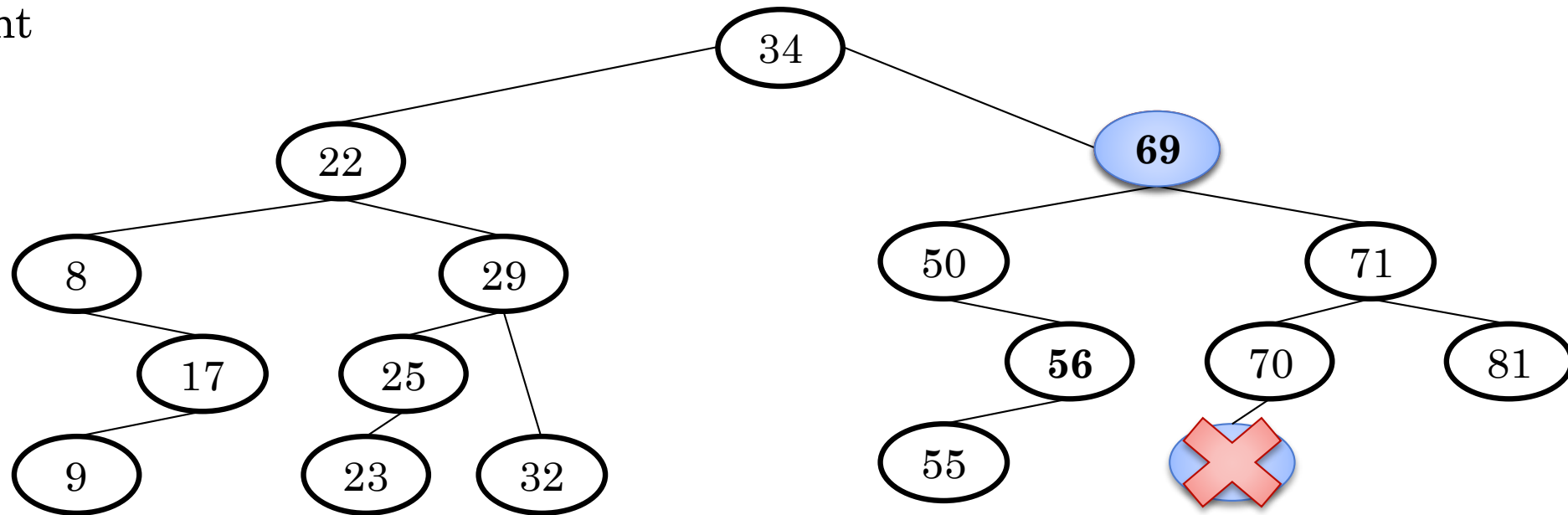
- Racine: 71
- La plus petite valeur : 69



II. OPÉRATION DE SUPPRESSION

❖ Cas 3: Suppression d'un nœud avec deux fils

- **Etape 2:** on applique à nouveau la procédure de suppression qui est maintenant une feuille ou un nœud avec un seul fils.
- Ainsi, si on choisit d'échanger le nœud « 66 » avec son plus proche successeur « 69 », on obtient

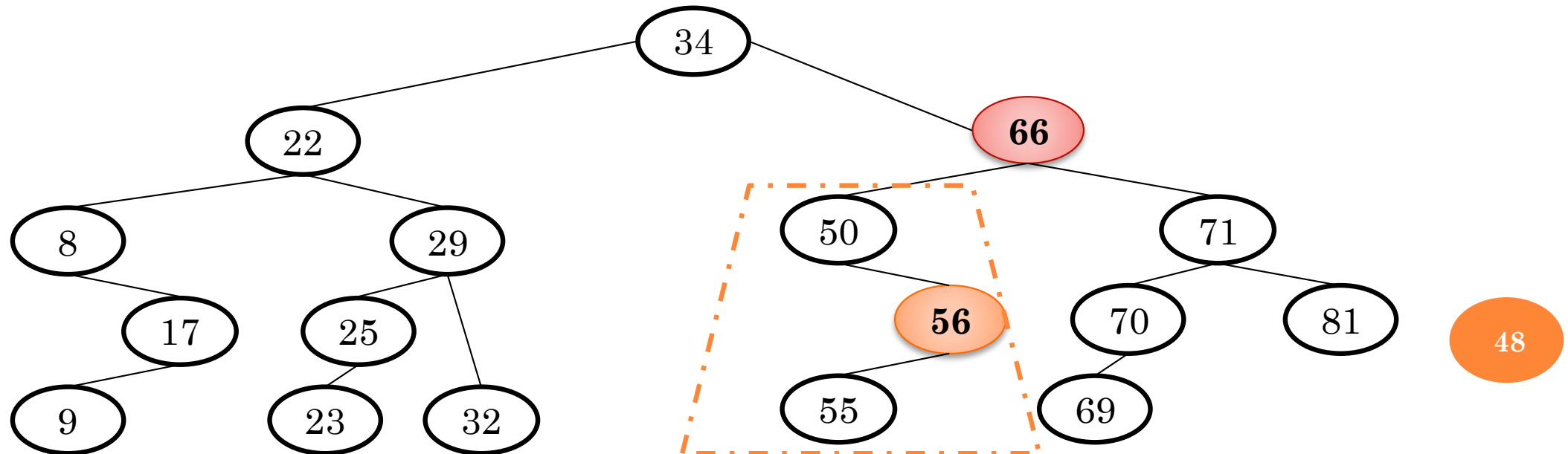


II. OPÉRATION DE SUPPRESSION

❖ Cas 3: Suppression d'un nœud avec deux fils

➤ **Etape 1 → Cas B:** On échange le nœud à supprimer avec son **plus proche prédécesseur** (le nœud le plus à droite ou le plus grand du sous-arbre gauche).

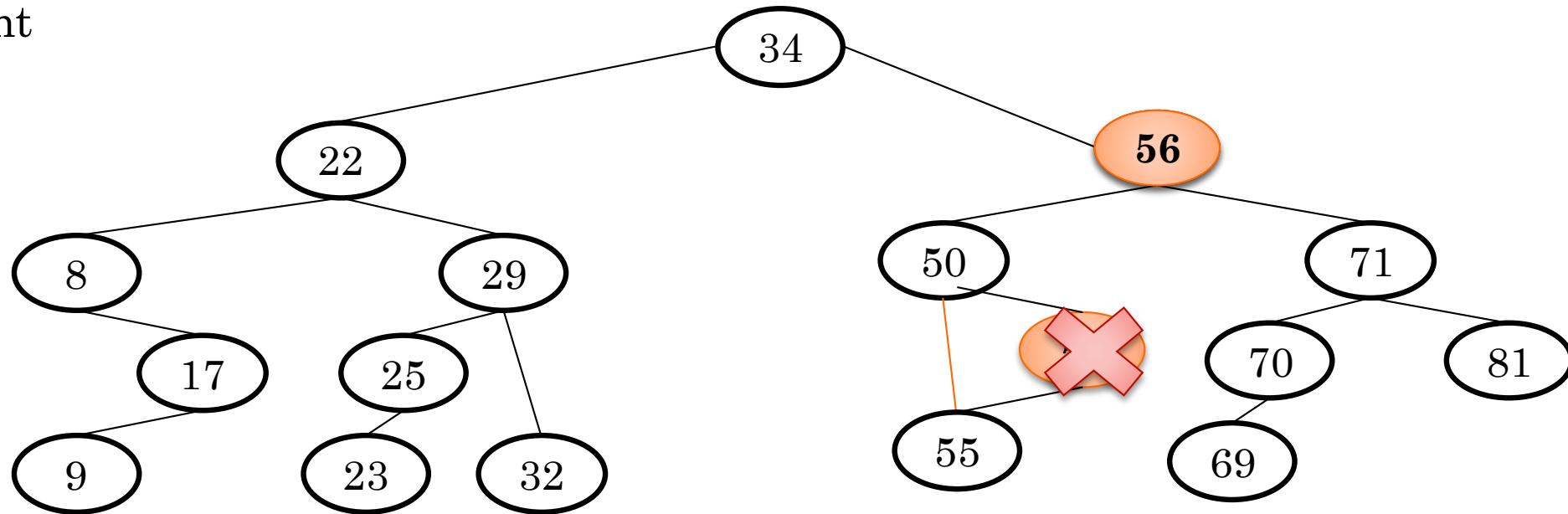
- Racine: 50
- La plus grande valeur : 56



II. OPÉRATION DE SUPPRESSION

❖ Cas 3: Suppression d'un nœud avec deux fils

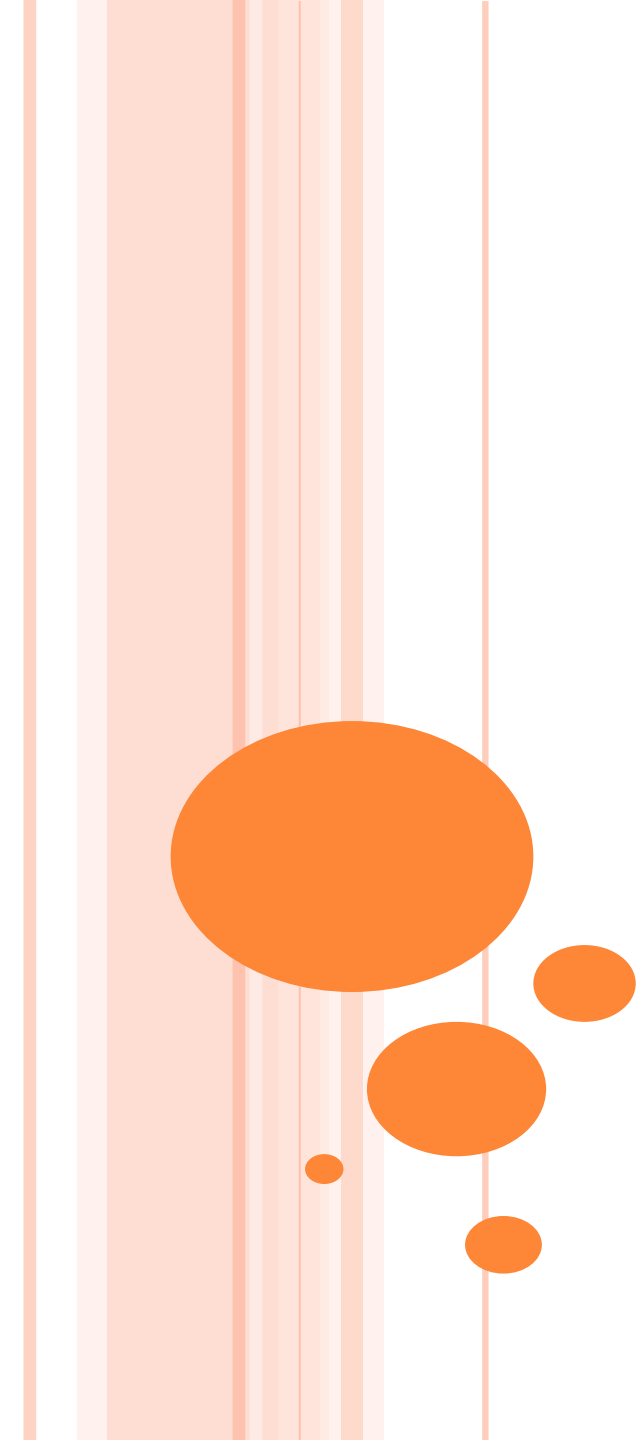
- Puis on applique à nouveau la procédure de suppression qui est maintenant une feuille ou un nœud avec un seul fils.
- Ainsi, si on choisit d'échanger le nœud « 66 » avec son plus proche prédécesseur « 56 », on obtient



II. OPÉRATION DE SUPPRESSION

- ❖ En conclusion, pour supprimer le nœud « i » d'un ARB, on rencontre une des situations suivantes :

Cas	FG (i)	FD (i)	Action
Feuille	Nil	Nil	Libérer le nœud « i »
Avec un fils	Nil	≠Nil	Chaîner le père au fils de « i » (FG(i) ou FD(i)) ensuite libérer le nœud « i »
	≠Nil	Nil	
Avec deux fils	≠Nil	≠Nil	1. Rechercher le plus proche prédécesseur ou successeur de « i », soit « P ». 2. Remplacer Info(i) par Info(P) 3. Supprimer le nœud « P »



PARTIE III:

ARBRES BINAIRES ÉQUILIBRÉS

PLAN DE LA PARTIE III

❑ AVL

- ❑ Introduction
- ❑ Définition
- ❑ Techniques d'équilibrage
- ❑ Opérations de Base: Recherche, Insertion et Suppression

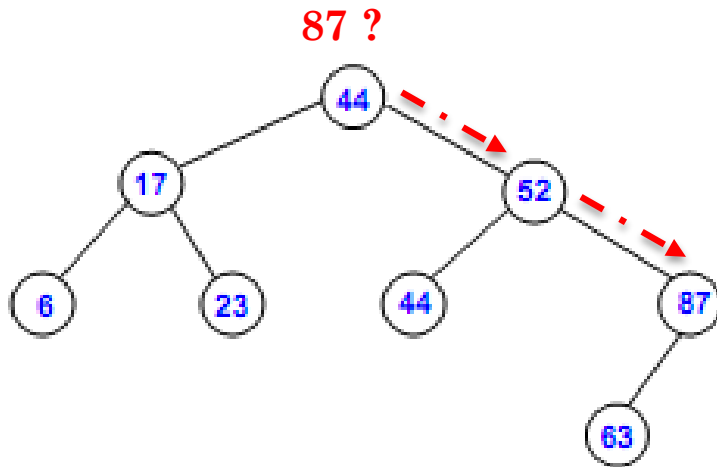
❑ TAS

- ❑ Définition
- ❑ Opérations de Base: Insertion, Recherche et Suppression
- ❑ Implémentation

III. AVL: INTRODUCTION

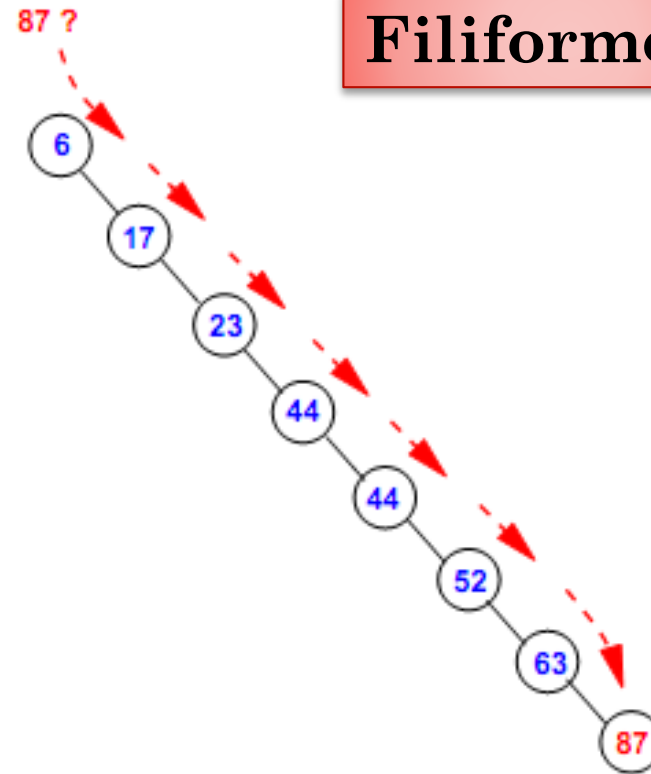
- ❖ Intérêt des ABR équilibrés est de diminuer la complexité temporelle des opérations de la recherche, de l'insertion et de la suppression dans un ABR quelconque.

ABR Equilibré



Nombre maximum de nœuds parcourus
 $\approx \log_2(n)$

Filiforme



Nombre maximum de nœuds parcourus
 $= n$

III. AVL: DÉFINITION

- ❖ Les arbres AVL ont été introduits par les finlandais Adelson- Velskii et Landis dans les années 60.
- ❖ Un arbre AVL est un ABR équilibré dont:
 - la différence de hauteur (ou profondeur) entre le sous-arbre gauche et le sous-arbre droit d'un nœud « R » diffère d'au plus 1.

$$| \text{Profondeur}(\text{FG}(\text{R})) - \text{Profondeur}(\text{FD}(\text{R})) | \leq 1$$

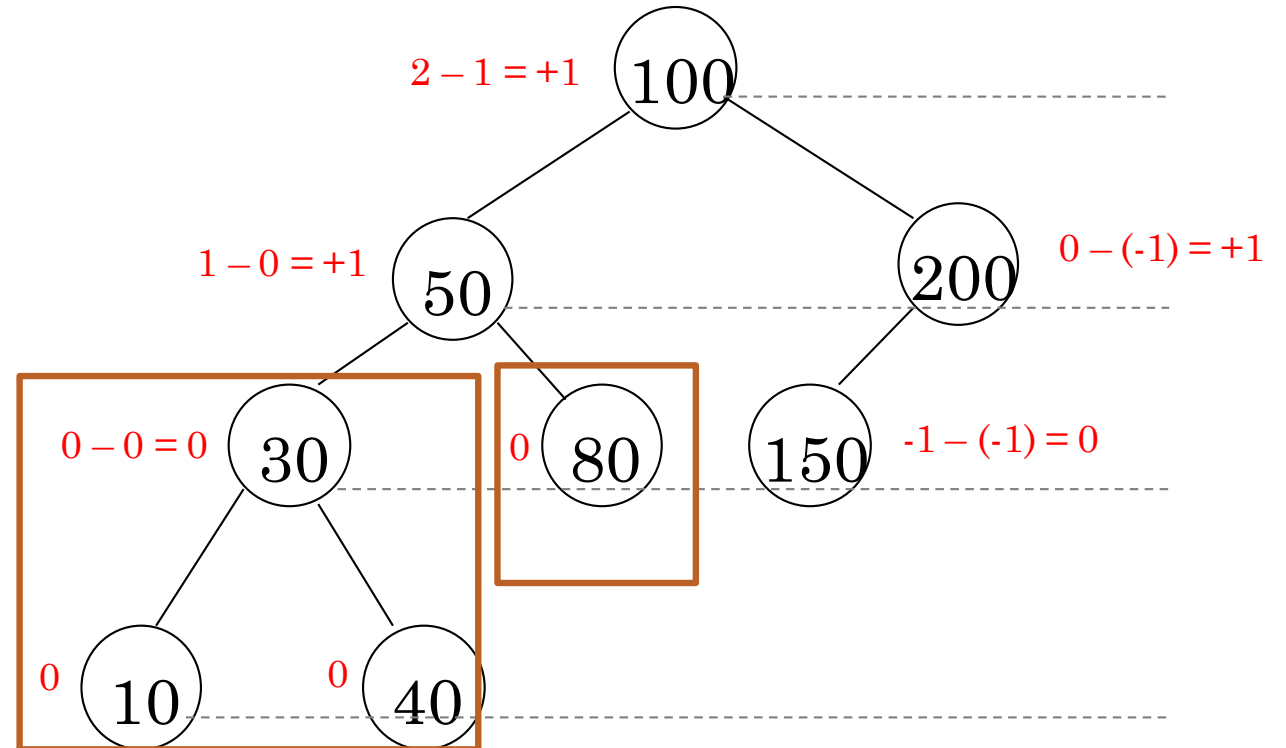
- les sous arbres gauches et droits d'un nœud sont des arbres AVL.
- ❖ Un champs supplémentaire est ajouté à tous les nœuds: c'est le **facteur de déséquilibre** (appelé aussi **facteur de balance**) qui est calculé après chaque insertion/suppression.

III. AVL: DÉFINITION

❖ **Exemple:** soit l'ABR suivant. Est-il un arbre AVL?

Rappelons:

- que l'arbre vide a la hauteur -1 ,
- Et qu'une feuille est un arbre de hauteur 0 .
- L'arbre vide et l'arbre réduit à une feuille, sont des arbres AVL



À vérifier pour chaque nœud R, on a:

$$| \text{Profondeur}(\text{FG}(\text{R})) - \text{Profondeur}(\text{FD}(\text{R})) | \leq 1$$

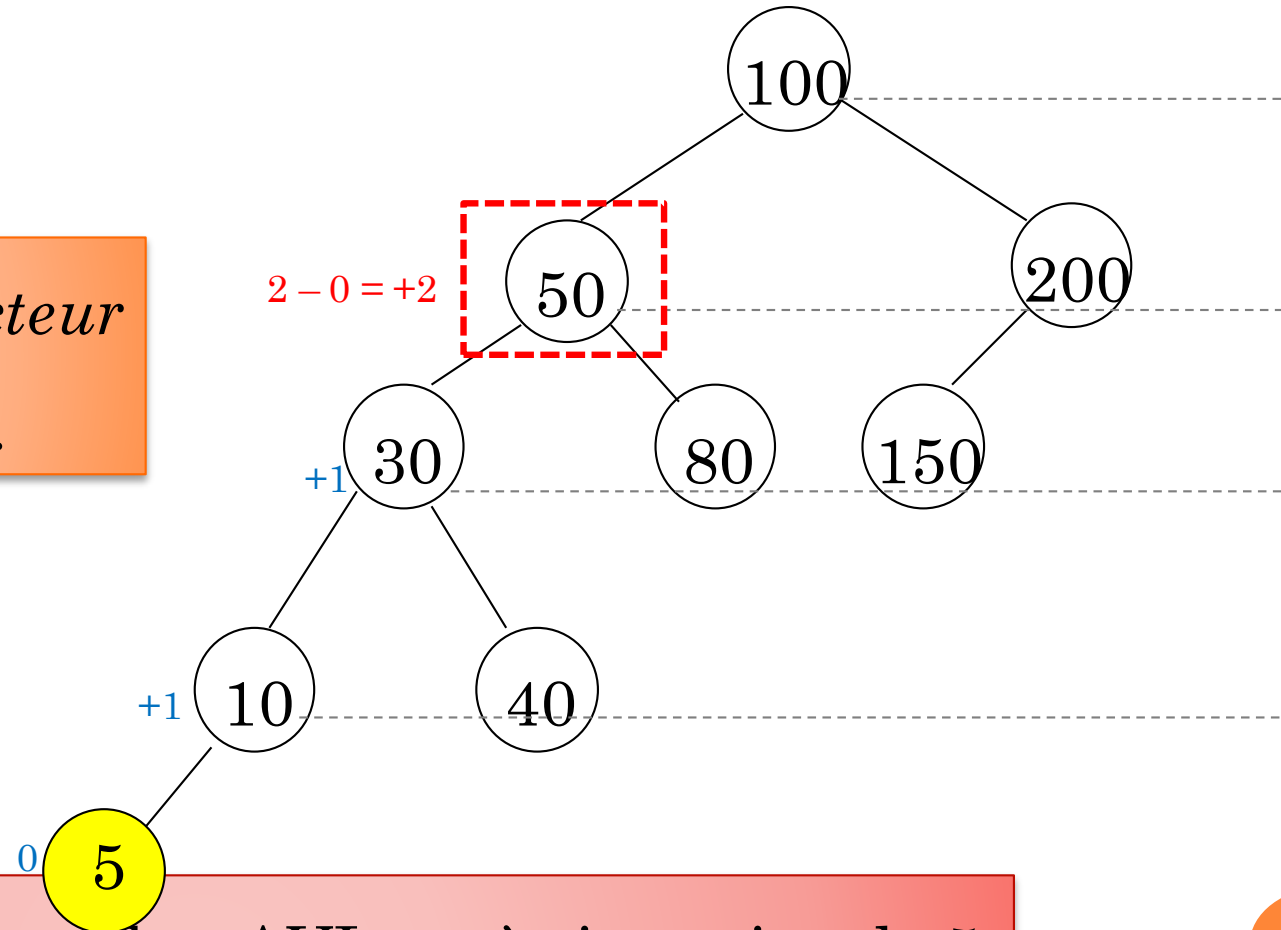
Cet arbre est un arbre AVL

III. AVL: DÉFINITION

❖ **Exemple:** Cet ABR est un arbre AVL avant insertion

➤ Insérer la valeur 5

Après insertion, recalculer le facteur de déséquilibre de chaque nœud.



Cet arbre n'est pas un arbre AVL après insertion de 5

➔ arbre déséquilibré

III. AVL: TECHNIQUES D'ÉQUILIBRAGE

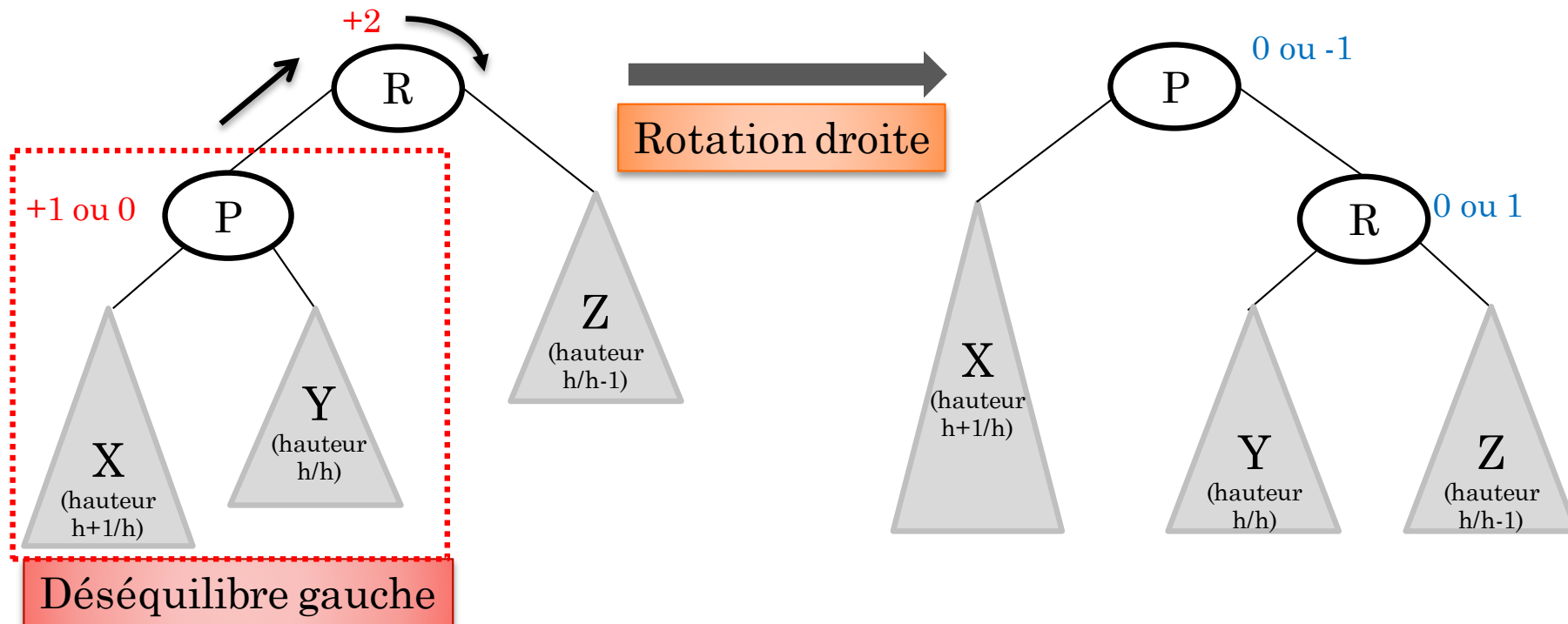
- ❖ L'opération d'équilibrage, appelée **rotation**, s'applique à tous les ABR dans le but de pouvoir les rééquilibrer:
 - On opère donc une **rotation gauche** lorsque l'arbre est «**déséquilibré à droite**», i.e. son sous-arbre droit est plus haut que son sous-arbre gauche.
 - On opère une **rotation droite** dans le cas contraire à savoir son sous-arbre gauche est plus haut que son sous-arbre droit.
- ❖ Les rotations préservent l'ordre des données d'un ABR (parcours inordre).

III. AVL: TECHNIQUES D'ÉQUILIBRAGE

❖ Rotation simple : Rotation droite

- Soit $A=(B, R, Z)$ un ABR tel que $B=(X, P, Y)$.
- La rotation droite est l'opération:

$$((X, P, Y), R, Z) \rightarrow (X, P, (Y, R, Z))$$

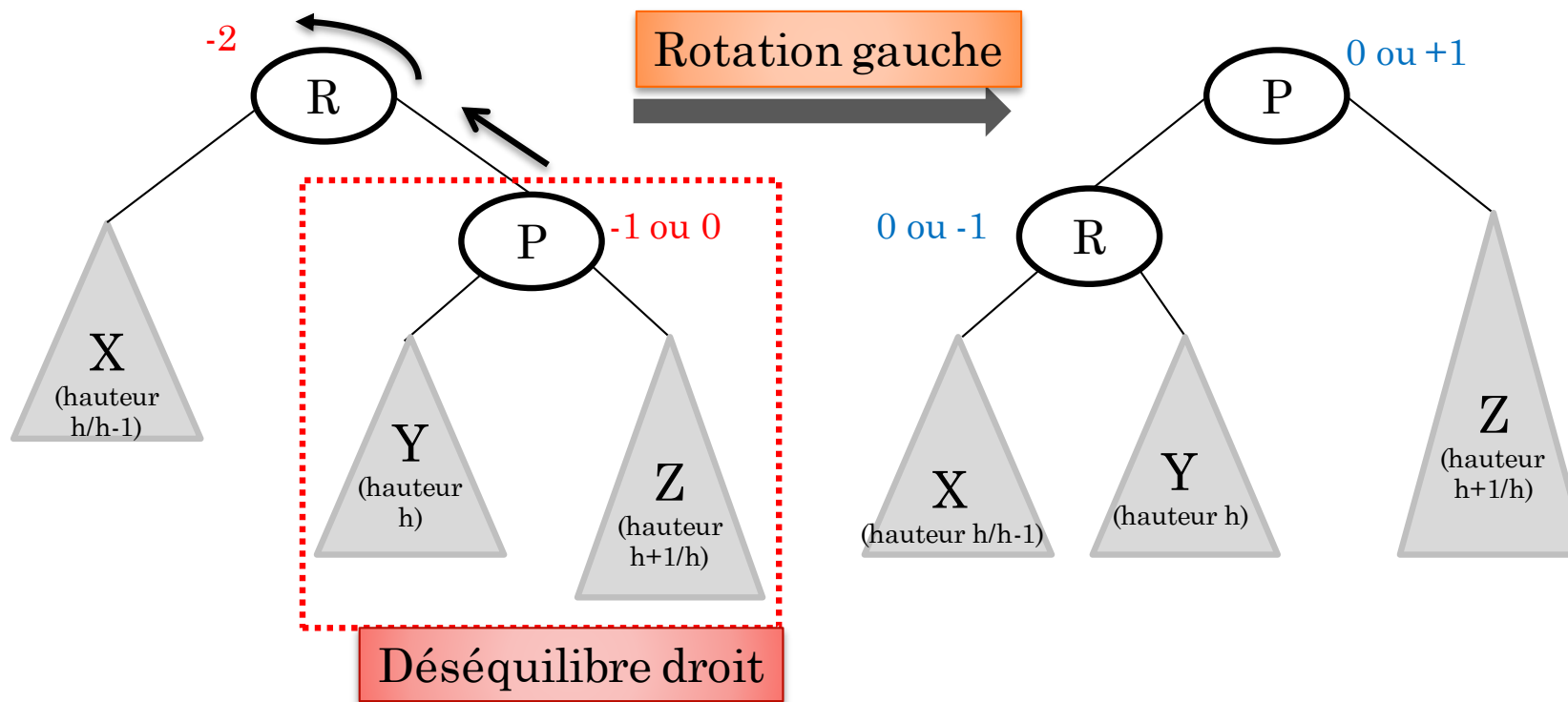


III. AVL: TECHNIQUES D'ÉQUILIBRAGE

❖ Rotation simple : Rotation gauche

- Soit $A=(X, R, B)$ un ABR tel que $B=(Y, P, Z)$.
- La rotation gauche est l'opération:

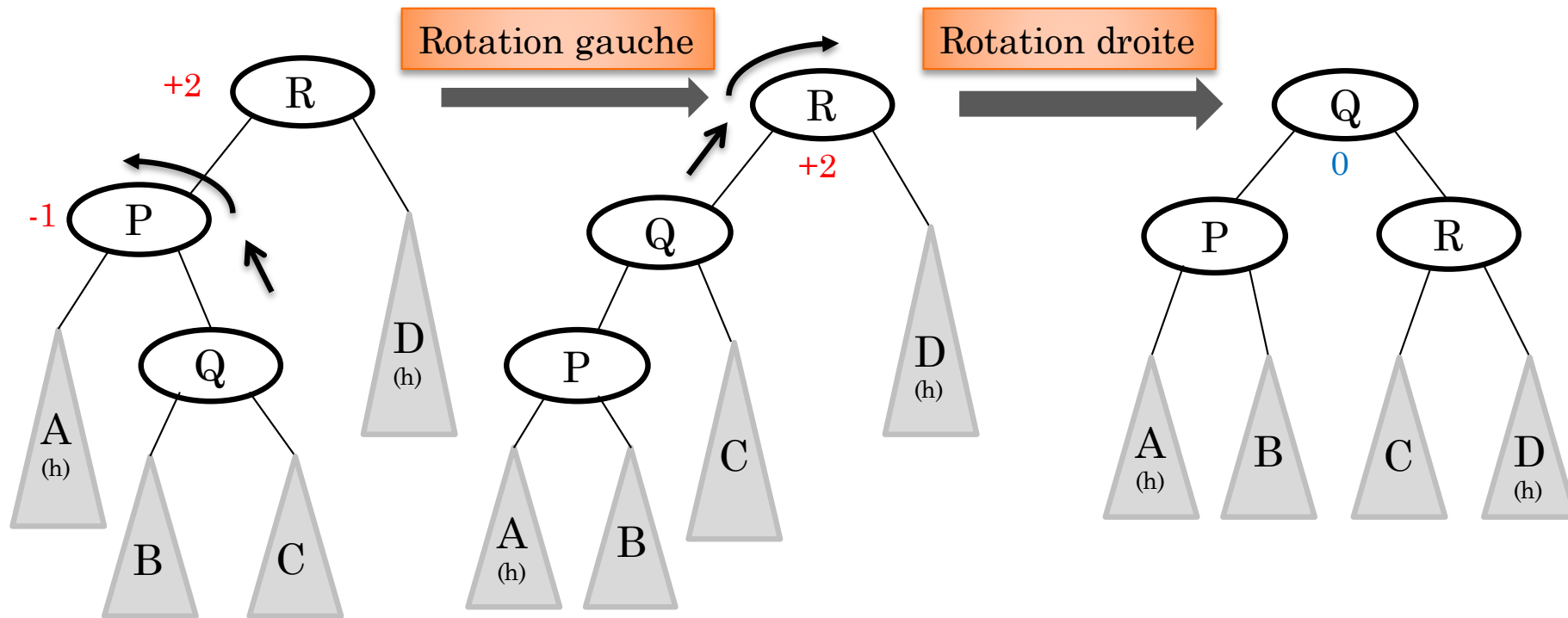
$$(X, R, (Y, P, Z)) \rightarrow ((X, R, Y), P, Z)$$



III. AVL: TECHNIQUES D'ÉQUILIBRAGE

❖ Rotation double : gauche-droite

- C'est une rotation gauche sur le sous arbre-gauche du nœud R suivie d'une rotation droite sur le nœud R

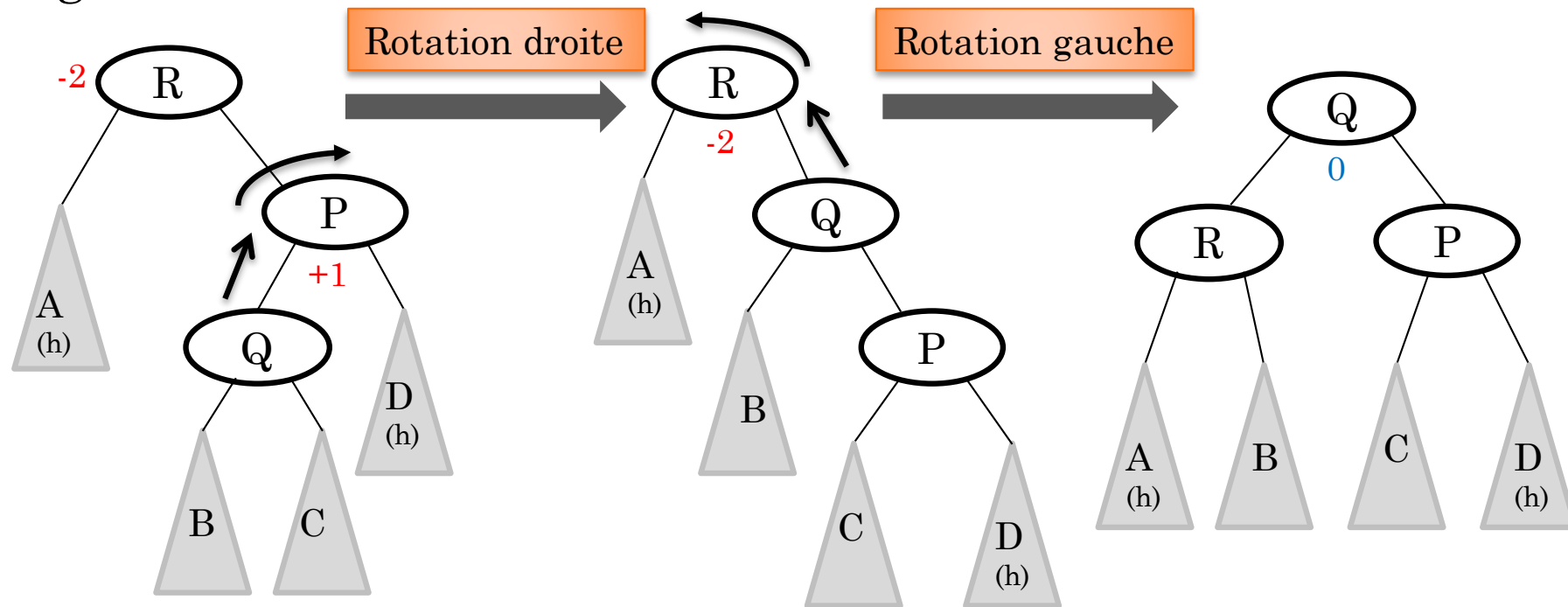


$((A, P, (B, Q, C)), R, D) \rightarrow (((A, P, B), Q, C), R, D) \rightarrow ((A, P, B), Q, (C, R, D))$

III. AVL: TECHNIQUES D'ÉQUILIBRAGE

❖ Rotation double : droite-gauche

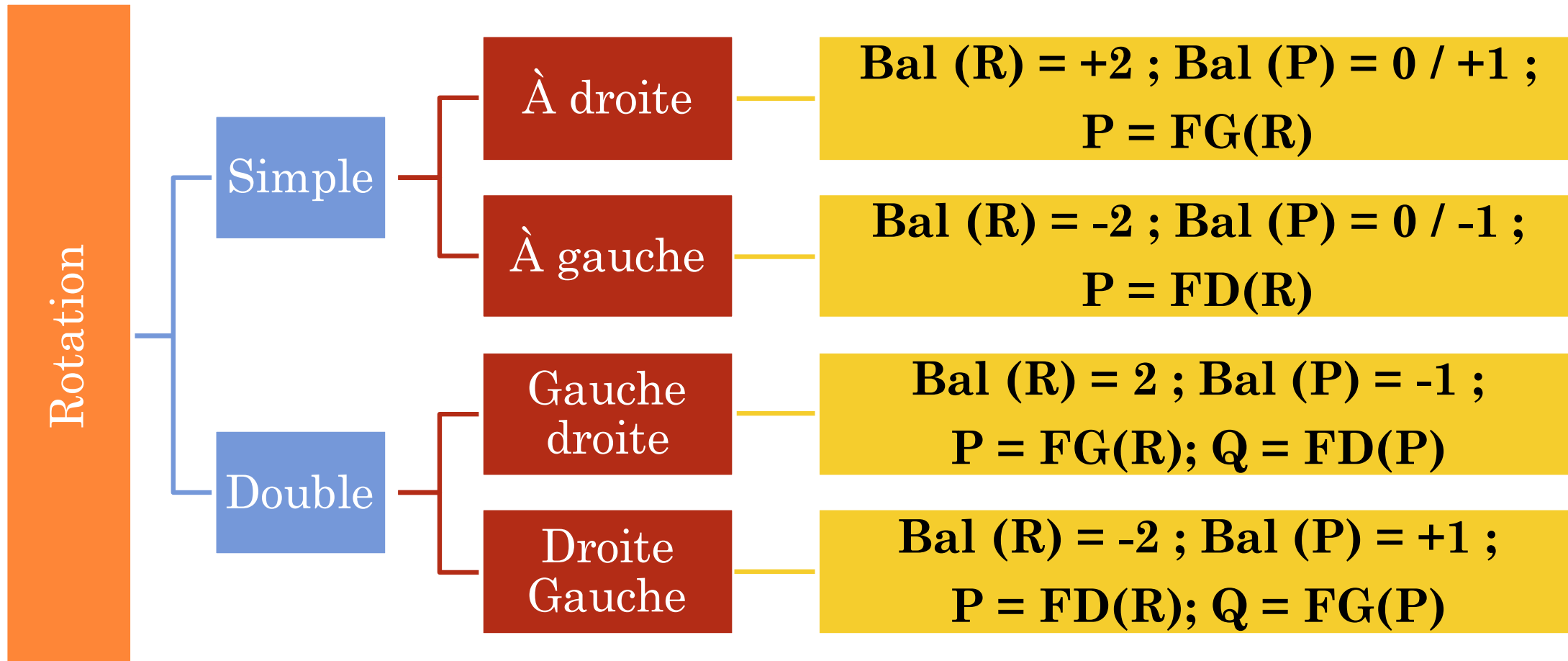
- C'est une rotation droite sur le sous arbre-droit du nœud R suivie d'une rotation gauche sur le nœud R



$(A, R, ((B, Q, C), P, D)) \rightarrow (A, R, (B, Q, (C, P, D))) \rightarrow ((A, R, B), Q, (C, P, D))$

III. AVL: TECHNIQUES D'ÉQUILIBRAGE

❖ En résumé, il y a 4 types de rotation



III. AVL: OPERATIONS DE BASE

- ❖ La **recherche** est identique à celui des ABR car les arbres AVL sont avant tout des ABR équilibrés.
- ❖ L'**insertion** d'un élément dans un arbre AVL peut provoquer un déséquilibre. Donc, pour rétablir l'équilibre (rééquilibrer) de l'arbre après une insertion, une seule rotation (simple ou double) suffit.
- ❖ La **suppression** d'un élément dans un arbre AVL peut provoquer un déséquilibre. Donc pour rétablir l'équilibre (rééquilibrer) de l'arbre après une suppression, il faut faire entre 1 et h rotations (h est la hauteur de l'arbre).

III. AVL: INSERTION

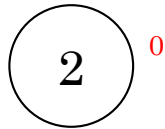
- ❖ L'ajout d'un nœud se fait toujours au niveau d'une feuille, puis on rééquilibre l'arbre AVL si l'insertion a déséquilibré l'arbre.
- ❖ Le déséquilibre est rencontré lorsque le facteur d'équilibrage d'un nœud de l'arbre égale à ± 2 .

III. AVL: INSERTION

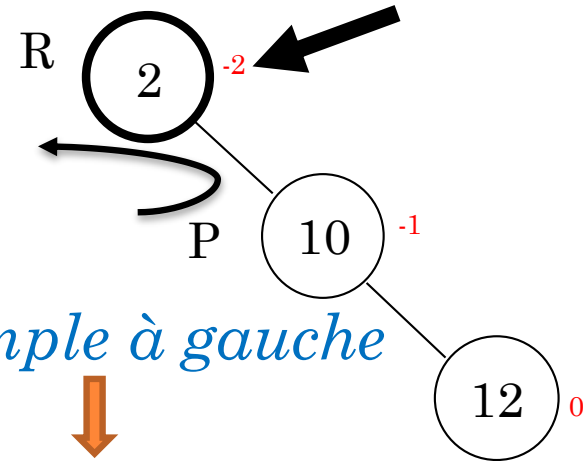
❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

(2 10 12 4 16 8 6 14 9 1 7)

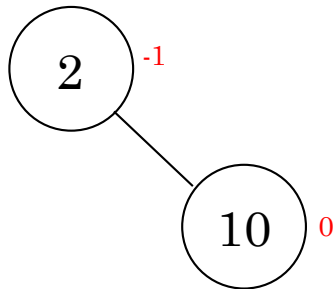
2 10 12 4 16 8 6 14



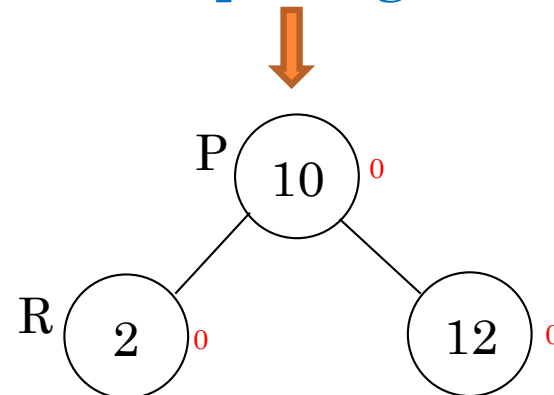
2 10 12 4 16 8 6 14



2 10 12 4 16 8 6 14



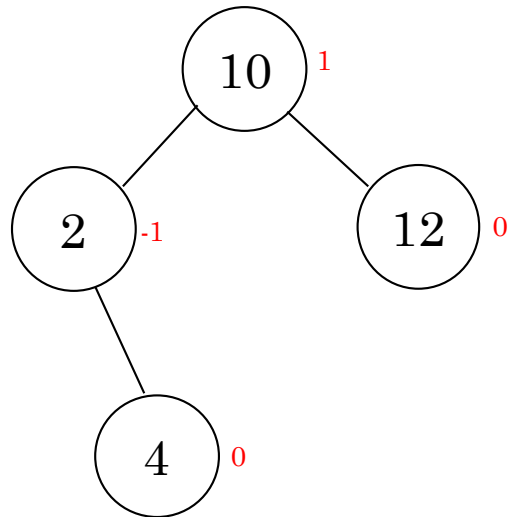
Rotation simple à gauche



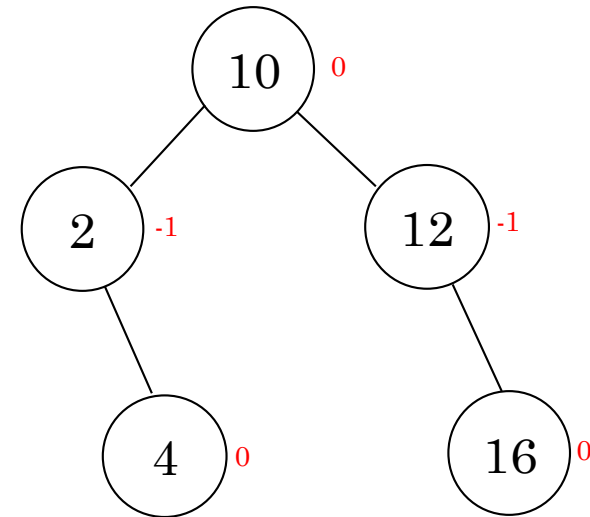
III. AVL: INSERTION

❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

2 10 12 4 16 8 6 14



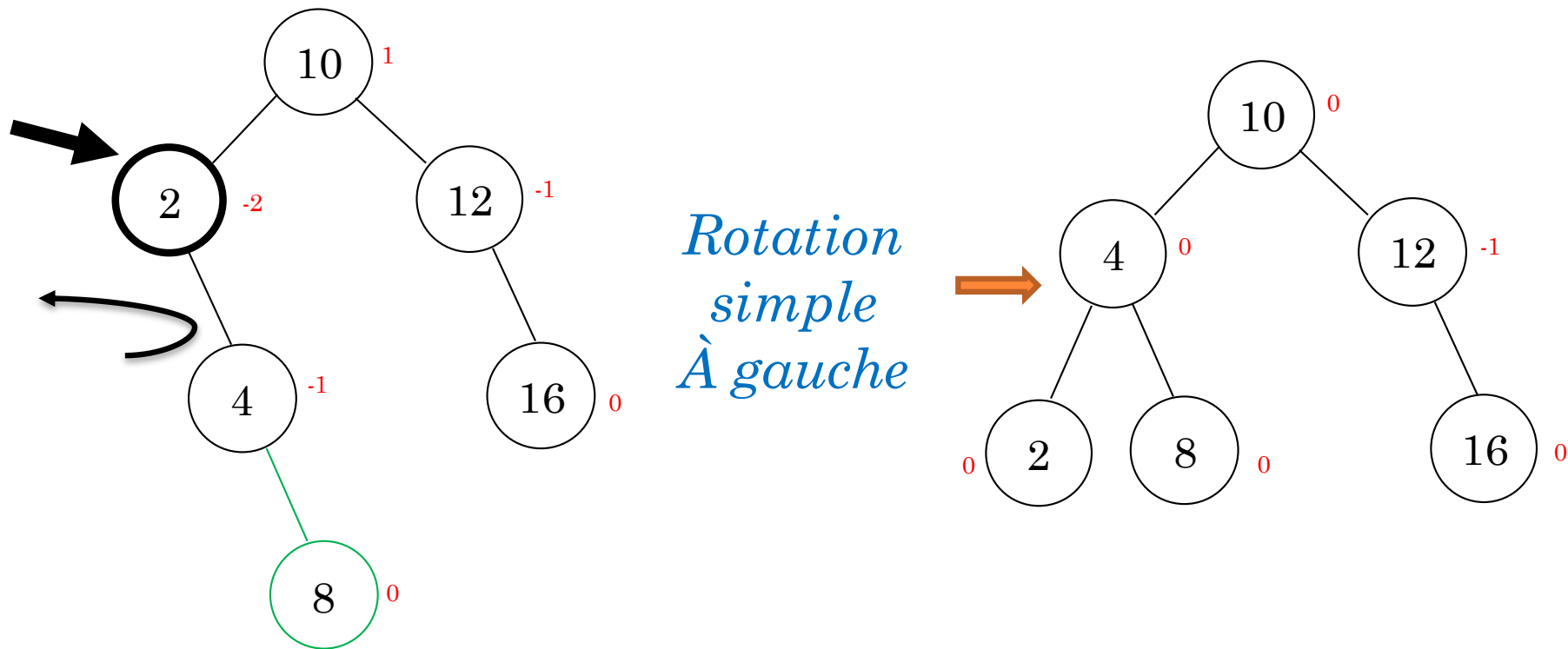
2 10 12 4 16 8 6 14



III. AVL: INSERTION

❖ **Exemple:** soit la série de nombres à insérer dans un arbre

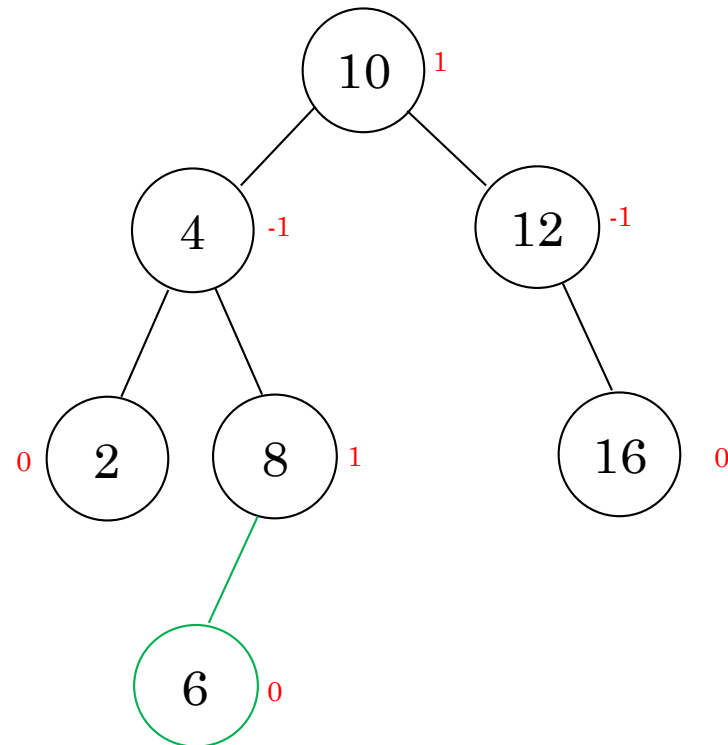
2 10 12 4 16 8 6 14 9 1 7



III. AVL: INSERTION

❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

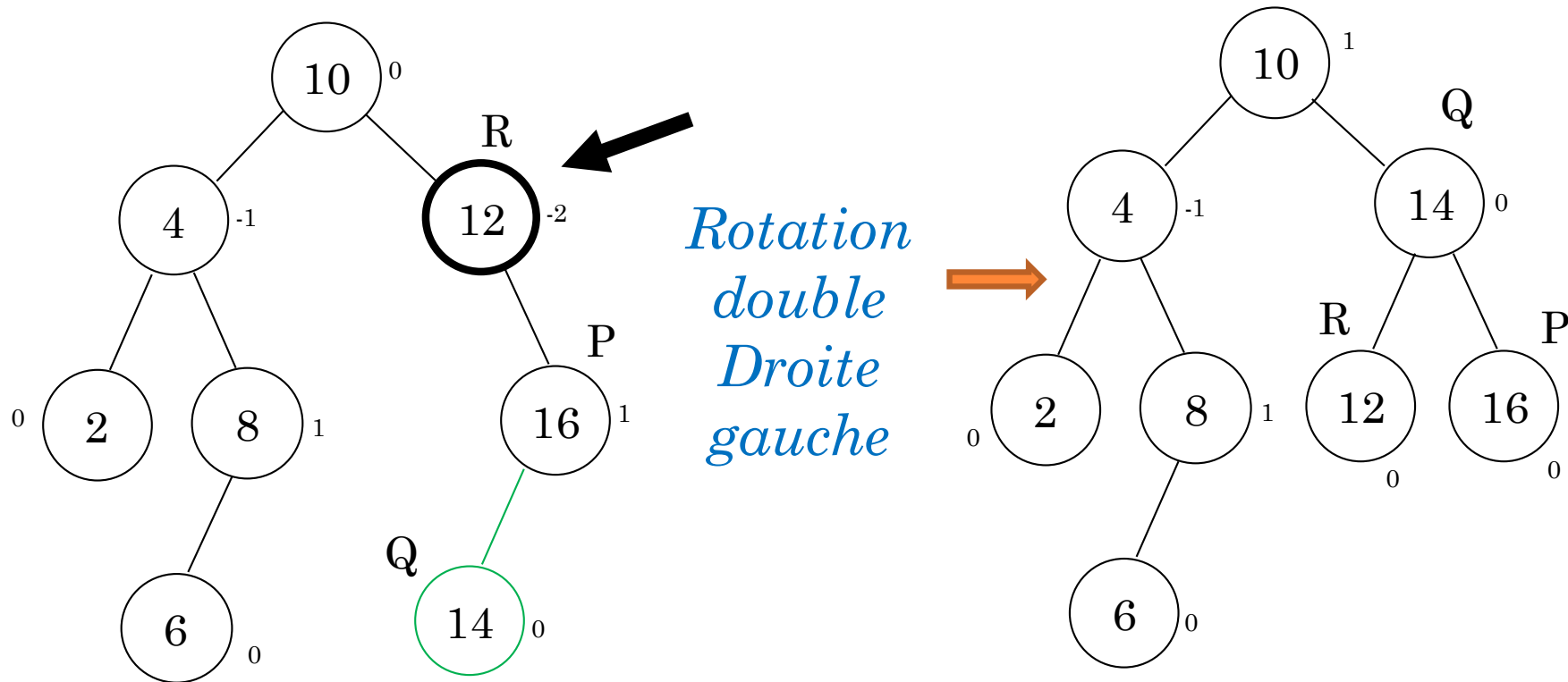
2 10 12 4 16 8 **6** 14 9 1 7



III. AVL: INSERTION

❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

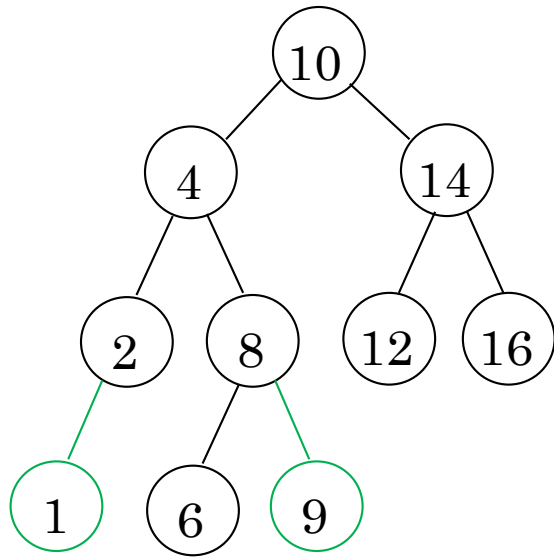
2 10 12 4 16 8 6 **14** 9 1 7



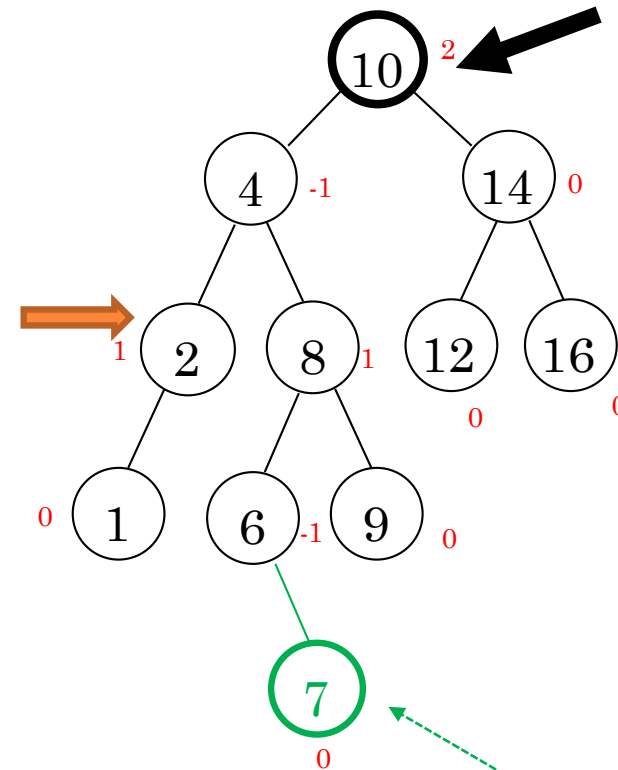
III. AVL: INSERTION

❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

2 10 12 4 16 8 6 14 9 1 7



*Insertion
du 7*

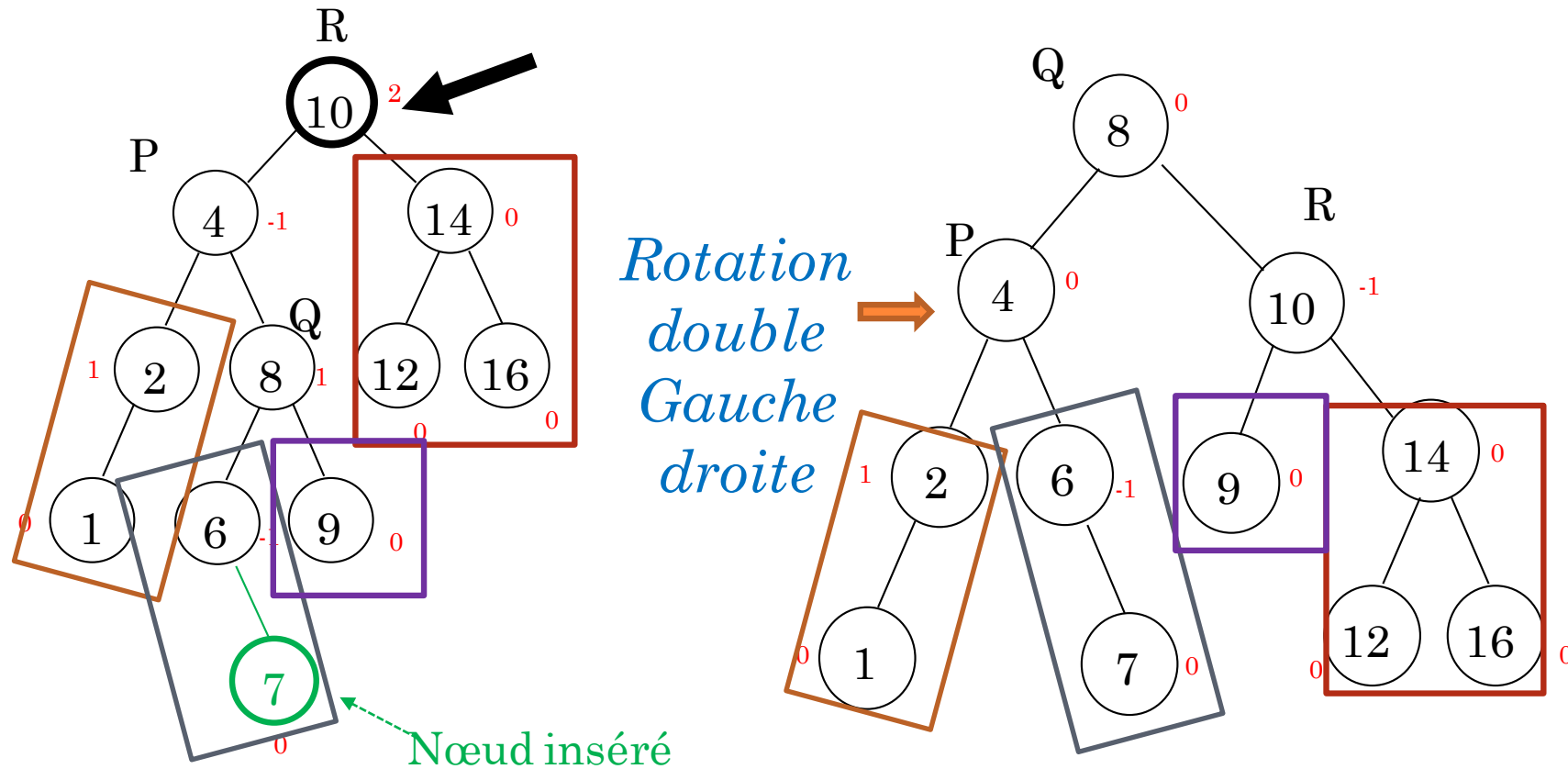


Nœud inséré

III. AVL: INSERTION

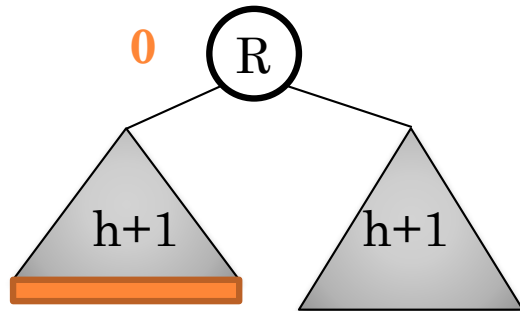
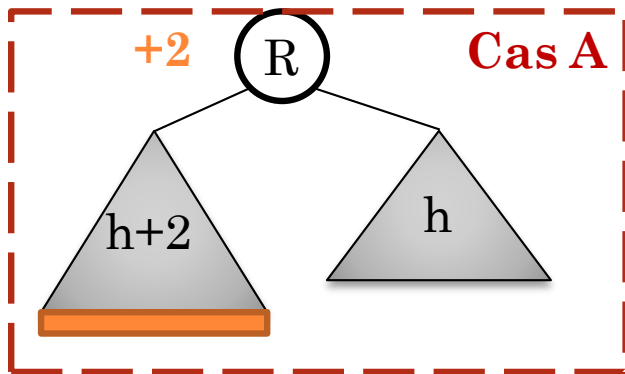
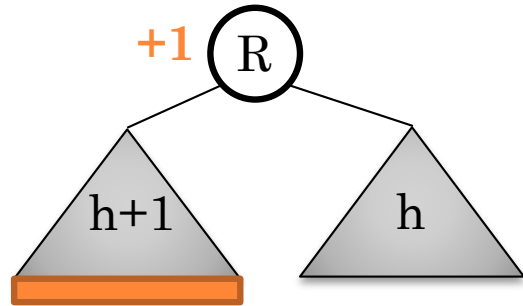
❖ **Exemple:** soit la série de nombres à insérer dans un arbre AVL

2 10 12 4 16 8 6 14 9 1 7

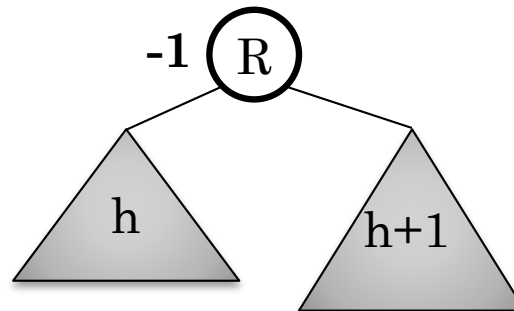
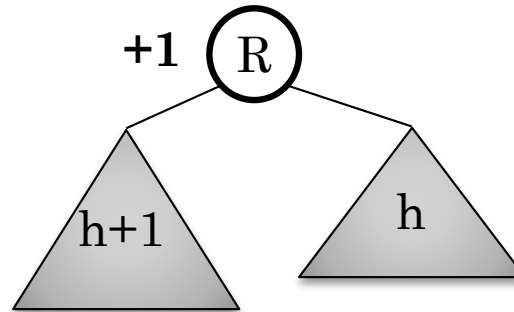
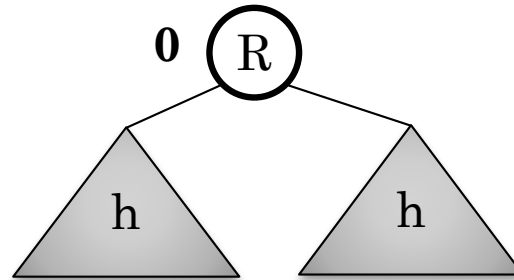


III. AVL: INSERTION

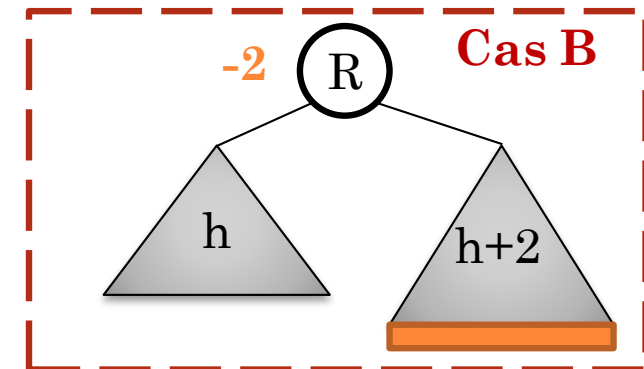
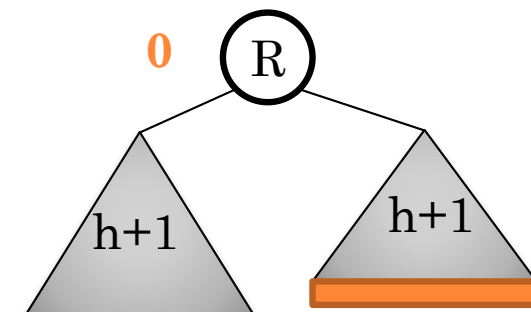
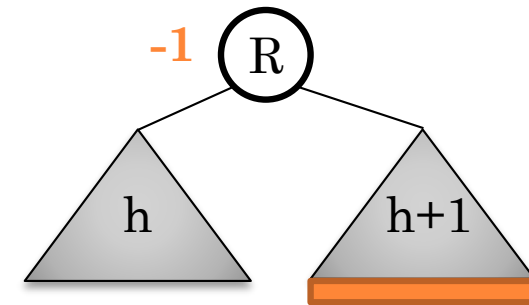
Après insertion à gauche



Avant



Après insertion à droite

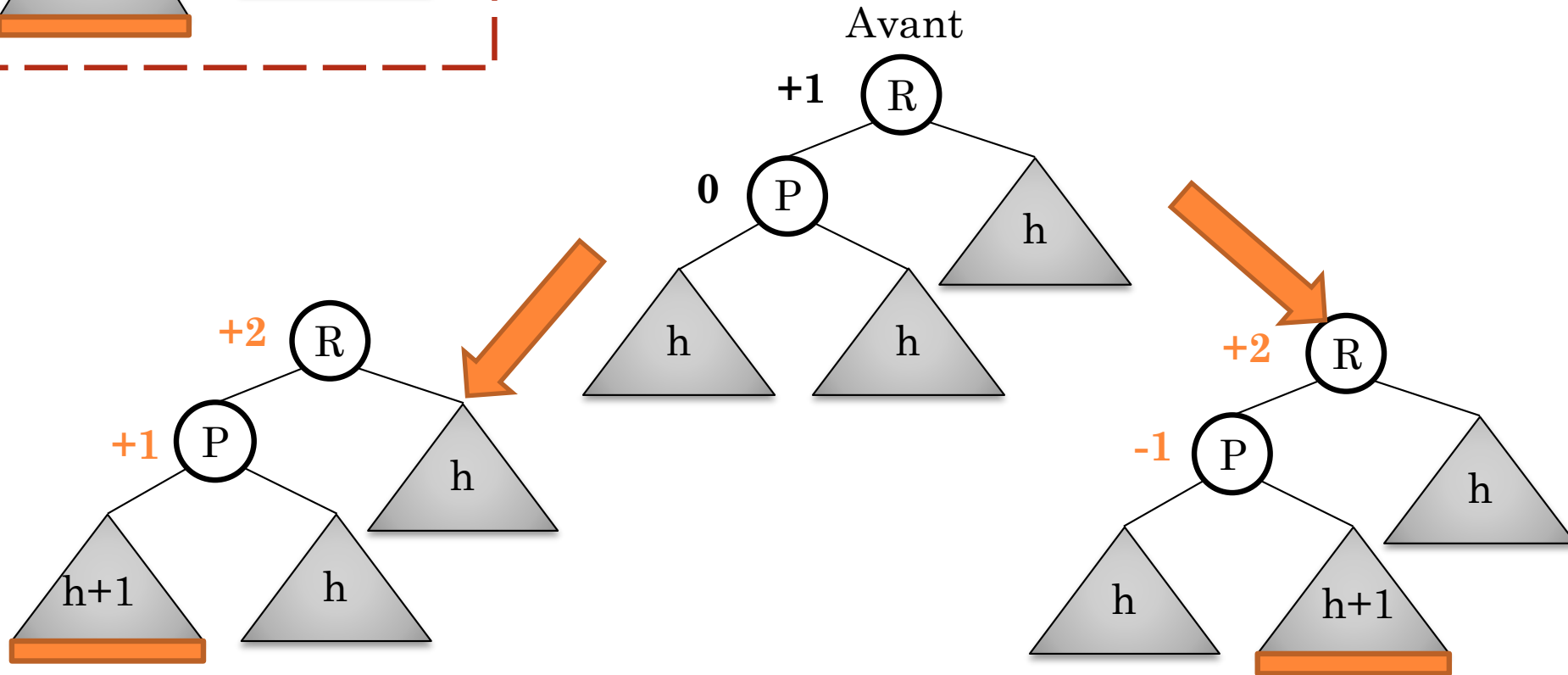
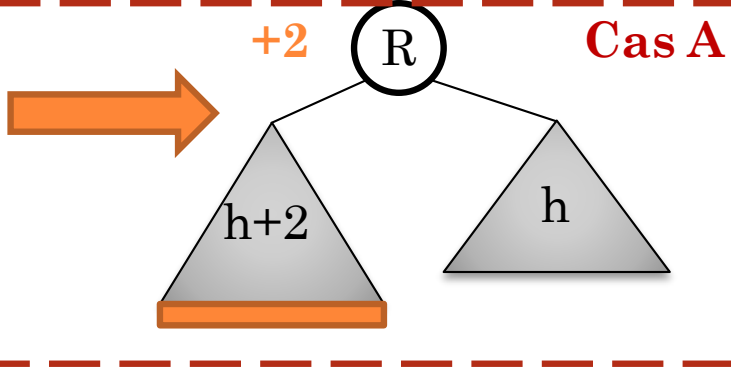


III. AVL: INSERTION

❖ Remarques:

- Après une insertion, seules les nœuds qui sont sur le chemin du point d'insertion à la racine sont susceptibles d'être déséquilibrés.
- **Cas A:** L'arbre devient non équilibré quand le nouveau nœud inséré est un descendant gauche d'un nœud qui avait un facteur d'équilibrage égal à 1
- **Cas B:** L'arbre devient non équilibré quand le nouveau nœud inséré est un descendant droit d'un nœud qui avait un facteur d'équilibrage égal à -1

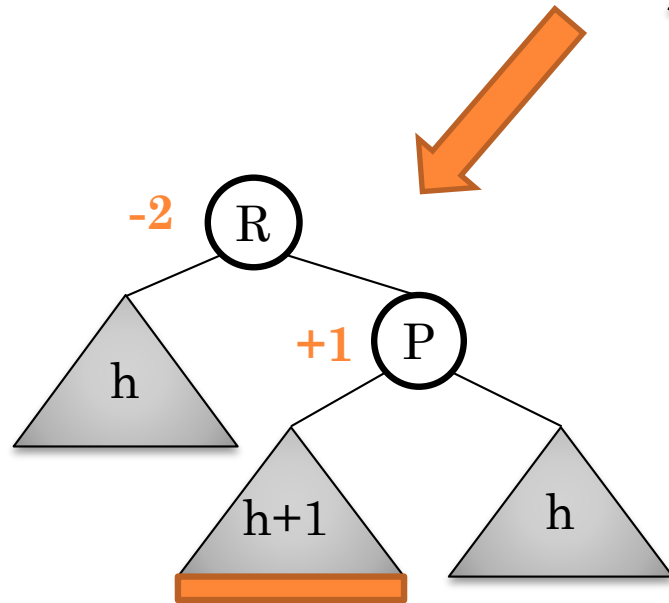
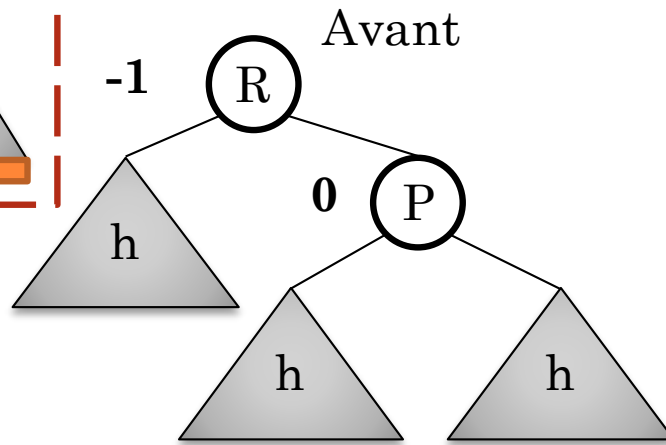
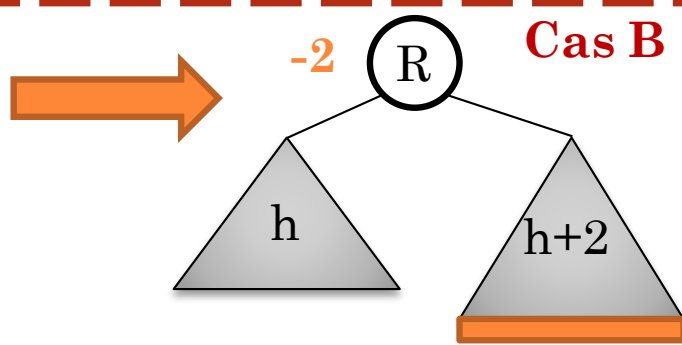
III. AVL: INSERTION



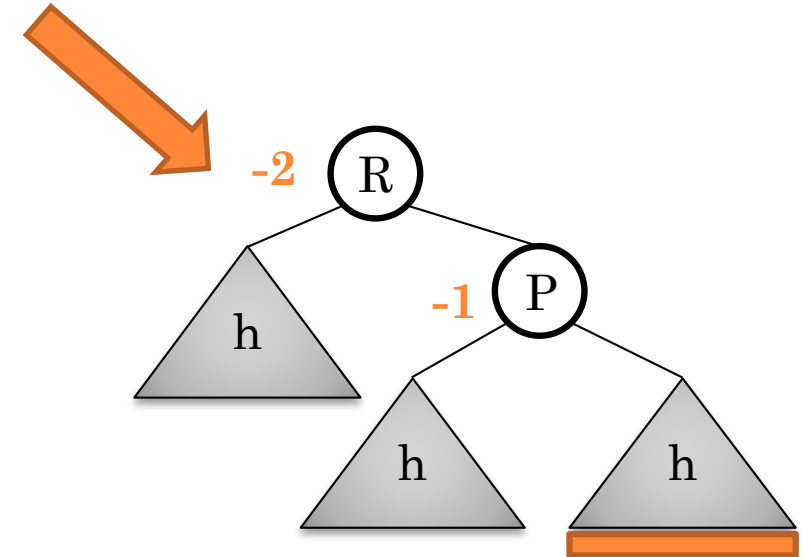
Si insertion dans le sous-arbre gauche du fils gauche alors **Rotation Simple à droite**

Si insertion dans le sous-arbre droit du fils gauche alors **Rotation Double Gauche-Droite**

III. AVL: INSERTION



Si insertion dans le sous-arbre gauche du fils droit alors **Rotation Double Droite-Gauche**



Si insertion dans le sous-arbre droit du fils droit alors **Rotation Simple à gauche**

III. AVL: SUPPRESSION

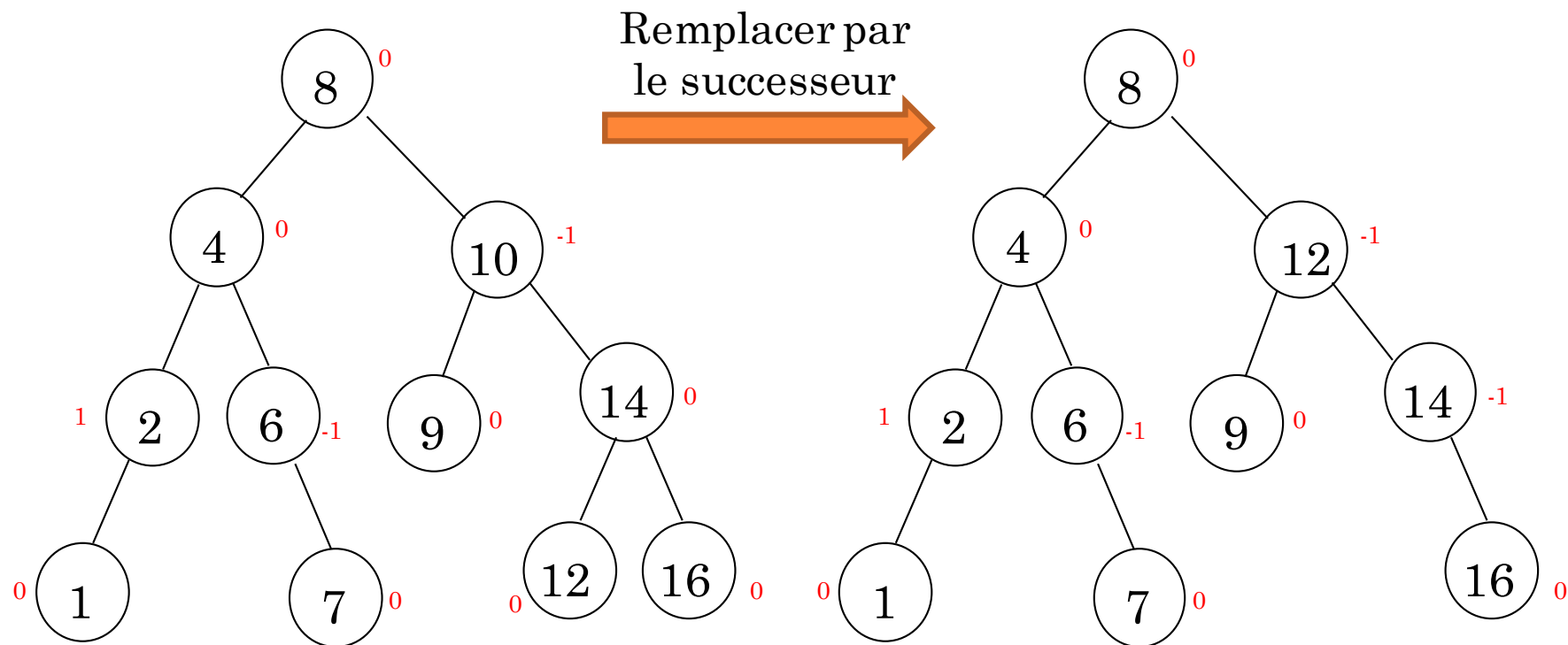
- ❖ Le principe de la suppression d'un élément dans un arbre AVL est le même que dans un ABR, c.à.d. recherche de l'élément à supprimer, suppression et remplacement, le cas échéant, par l'élément qui lui immédiatement inférieur ou supérieur.
- ❖ Après la première phase de la suppression, la hauteur de l'arbre est diminué de 1. Le problème est que cet arbre n'est plus forcément un arbre AVL. Il faut donc le rééquilibrer.

III. AVL: SUPPRESSION

- ❖ S'il y a déséquilibre, la rotation appliquée peut diminuer à son tour la hauteur de l'arbre et générer un nouveau déséquilibre. En fait les rotations peuvent s'enchaîner en cascade depuis l'élément supprimé jusqu'à la racine. Ainsi, on peut faire jusqu'à h (hauteur de l'arbre initial) rotations (simple ou double).

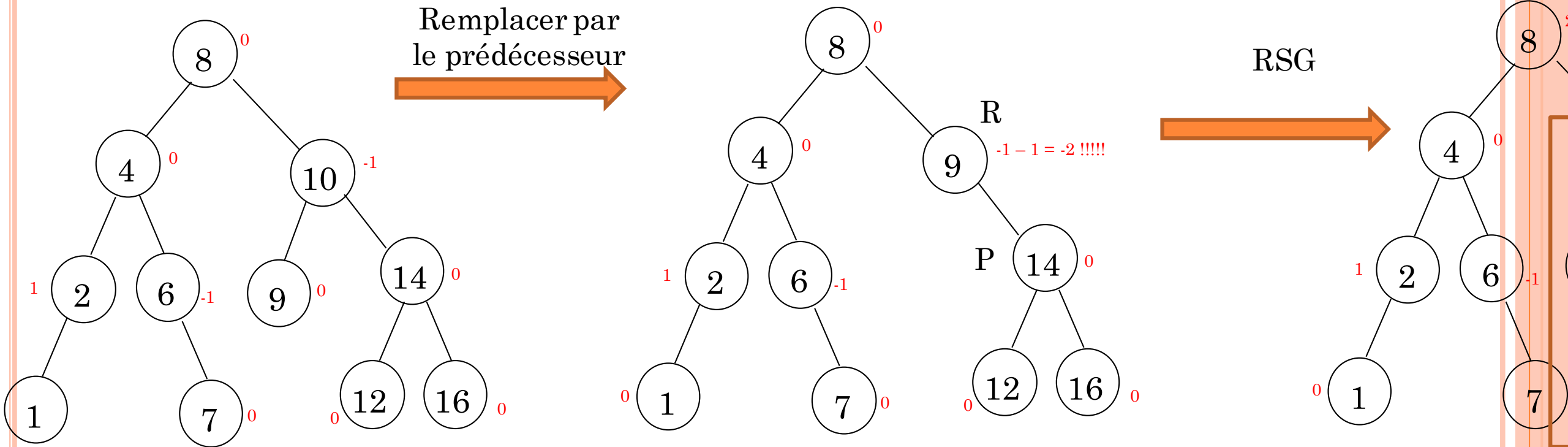
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 10:



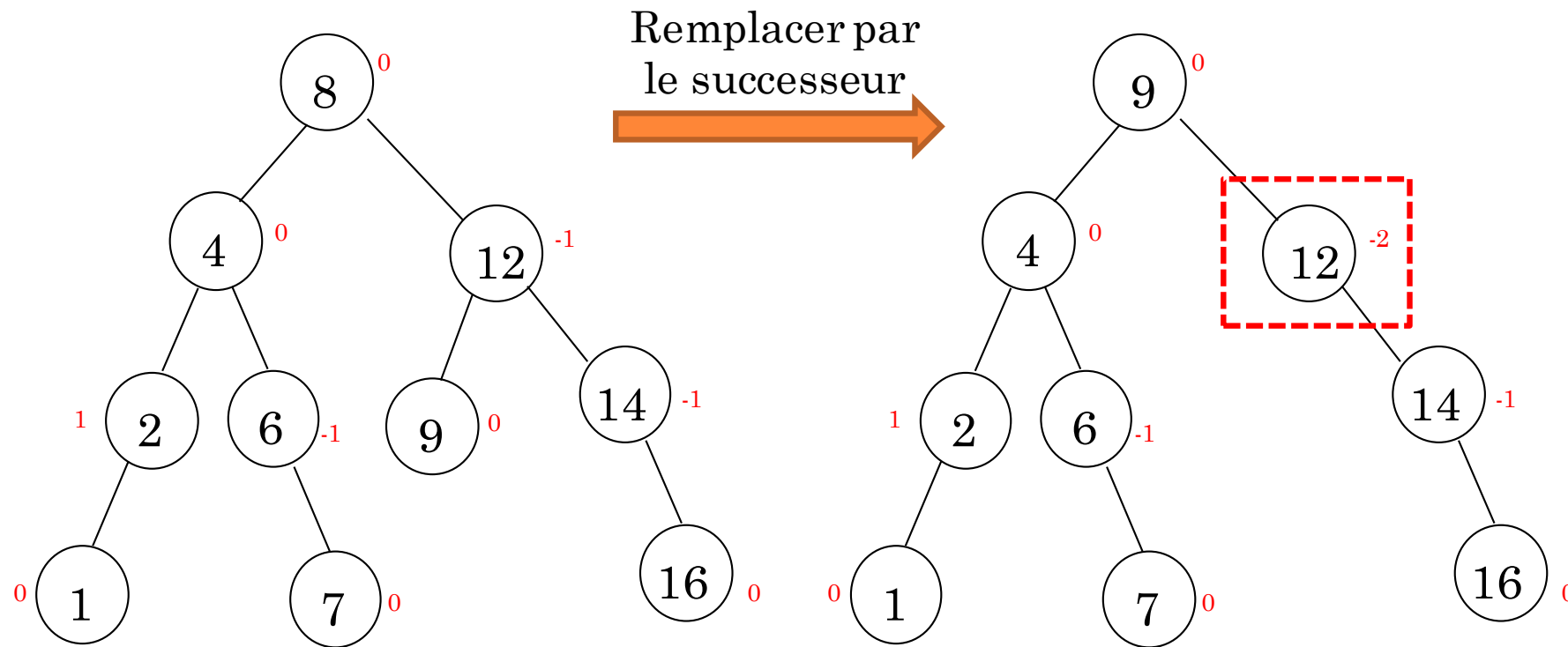
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 10:



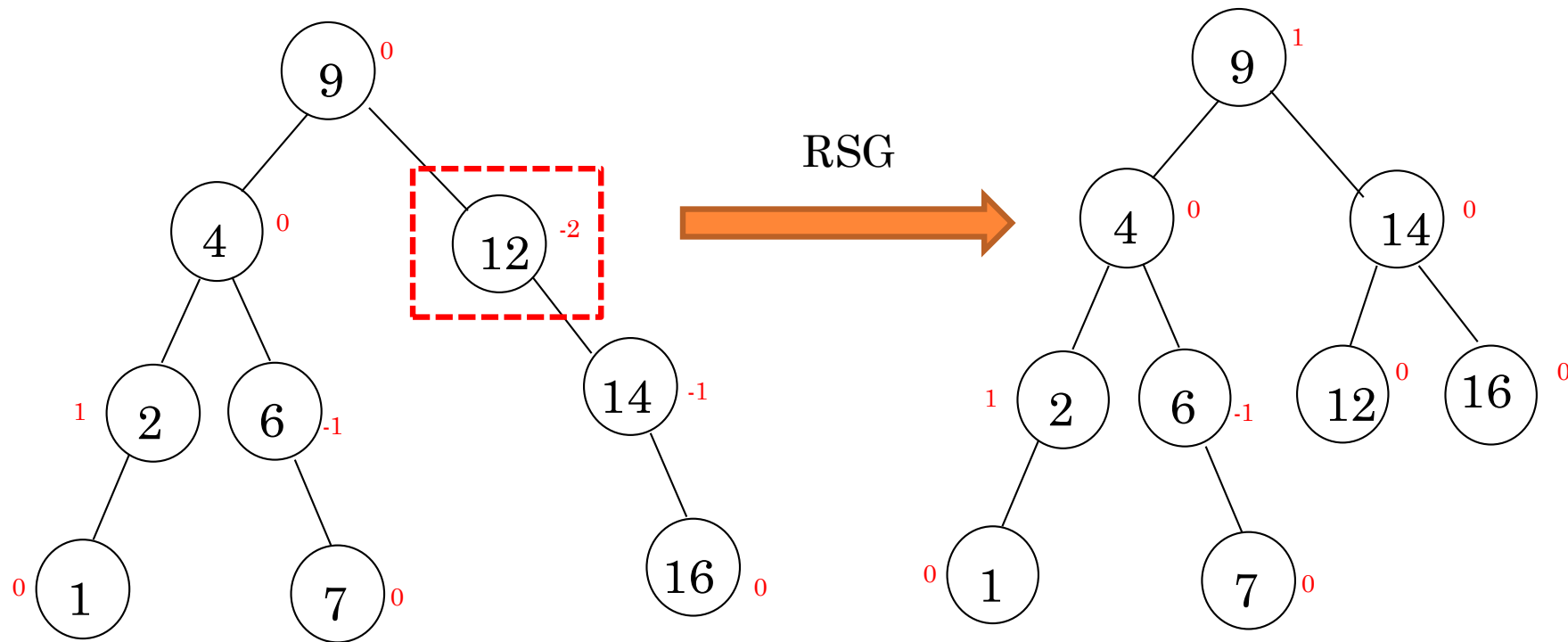
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 8:



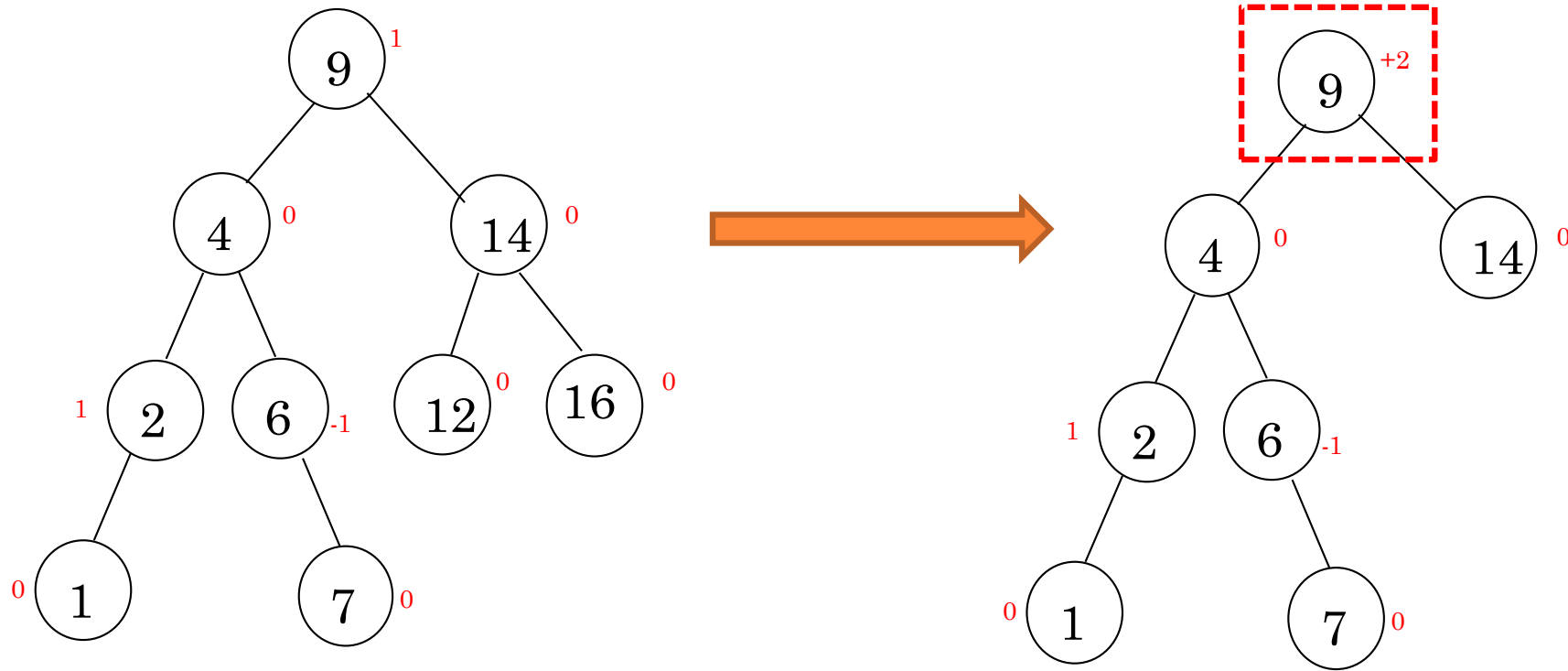
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 8:



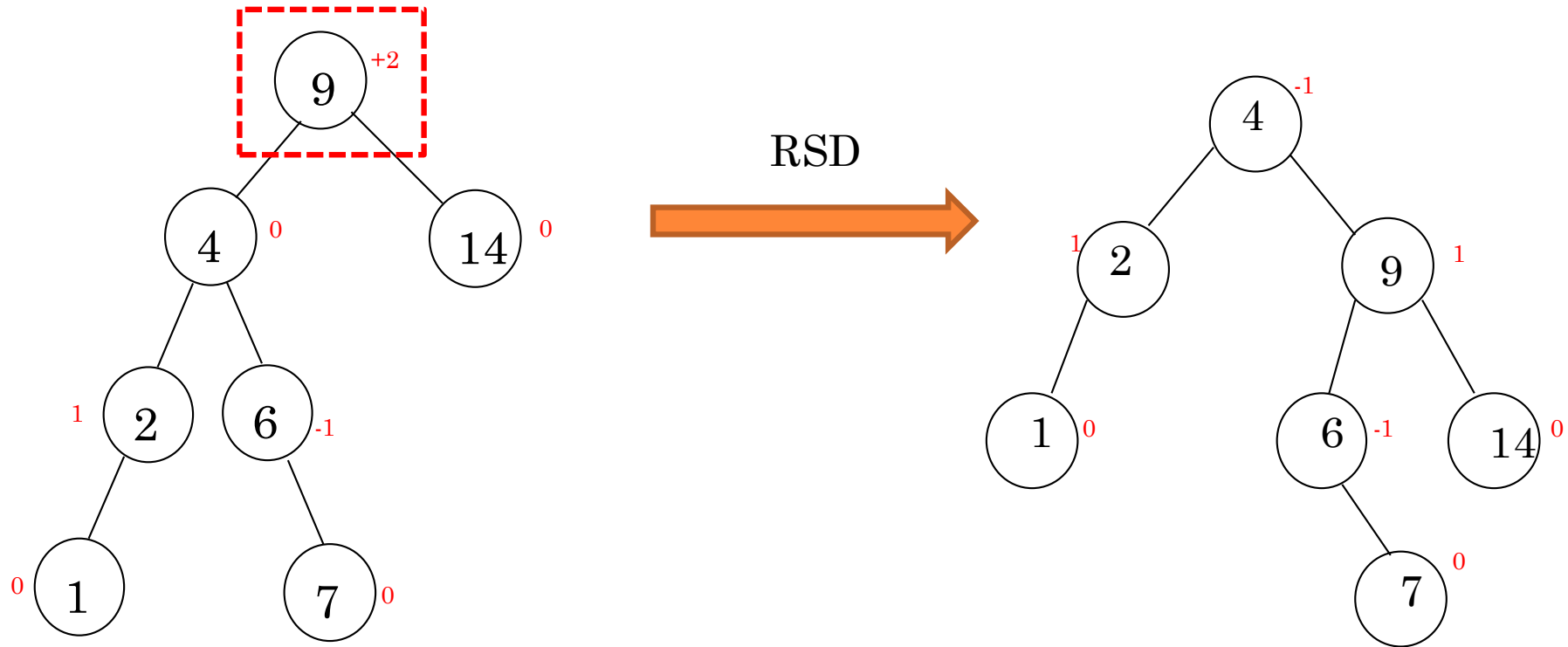
III. AVL: SUPPRESSION

- ❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 12 puis 16:



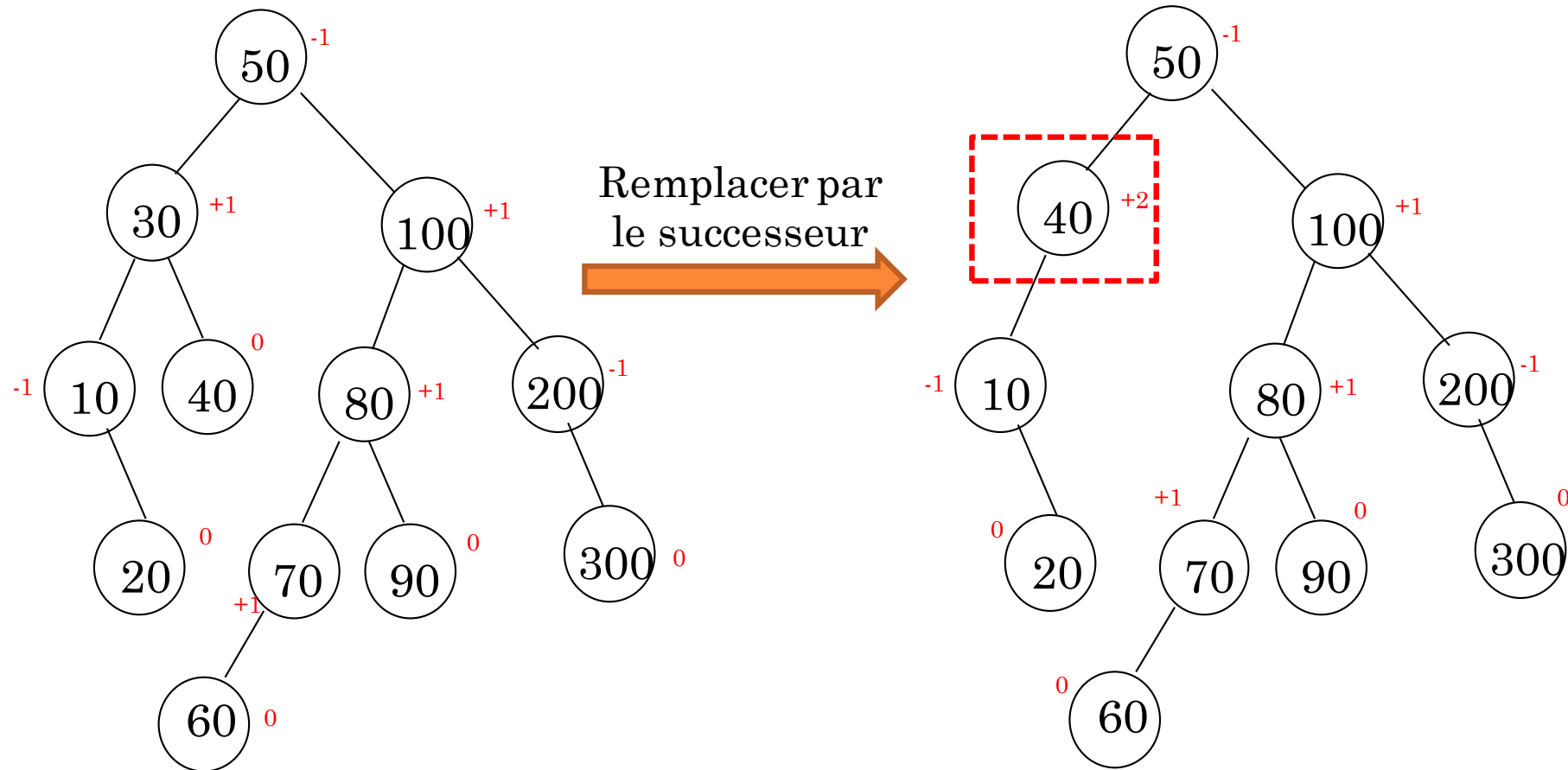
III. AVL: SUPPRESSION

- ❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 12 puis 16:



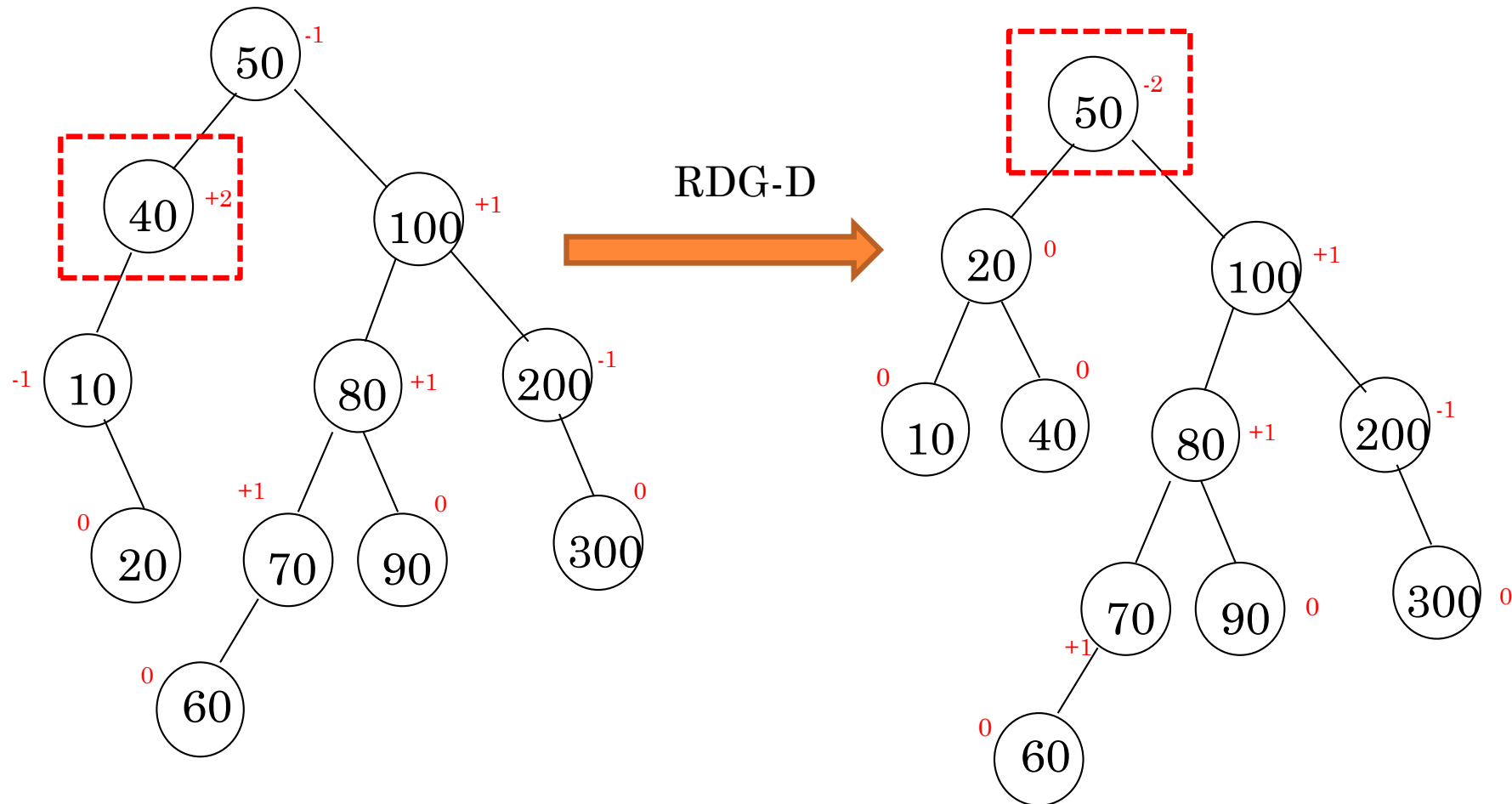
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 30:



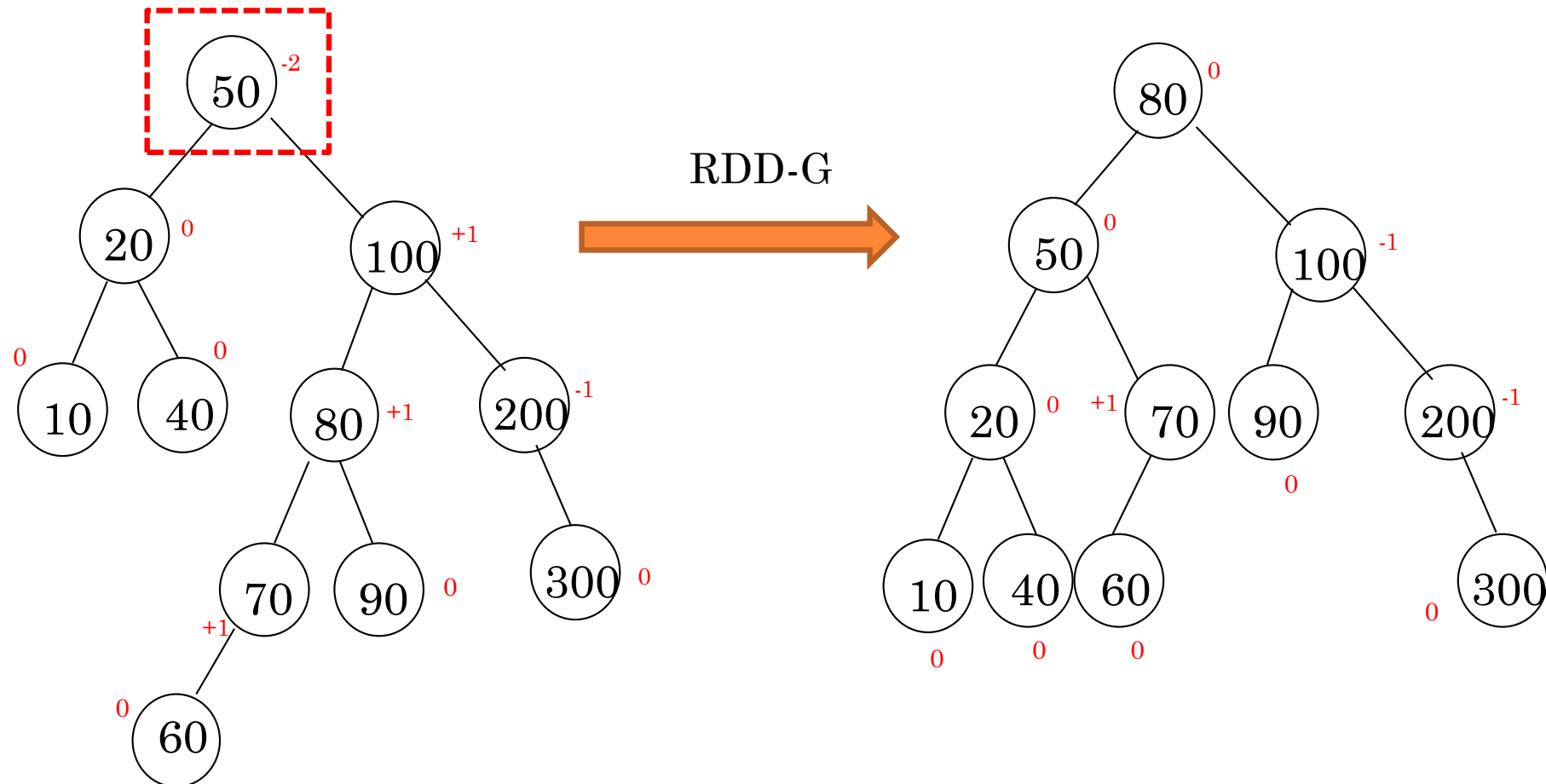
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 30:



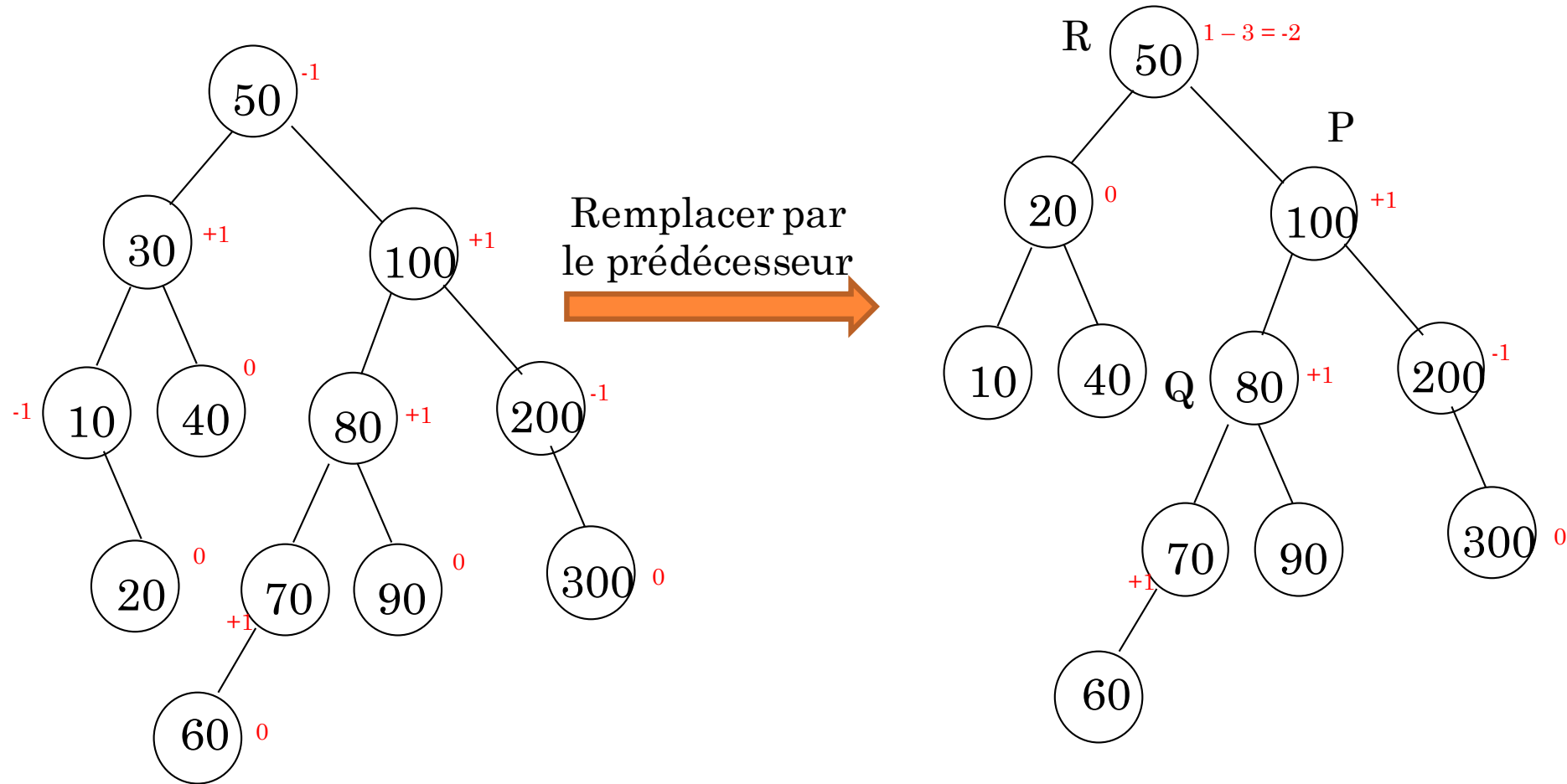
III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 30:



III. AVL: SUPPRESSION

❖ **Exemple:** soit l'arbre suivant. Donner le résultat après la suppression de 30:



PLAN DE LA PARTIE III

- AVL

- TAS

 - Définition

 - Opérations de Base: Insertion, Recherche et Suppression

 - Implémentation

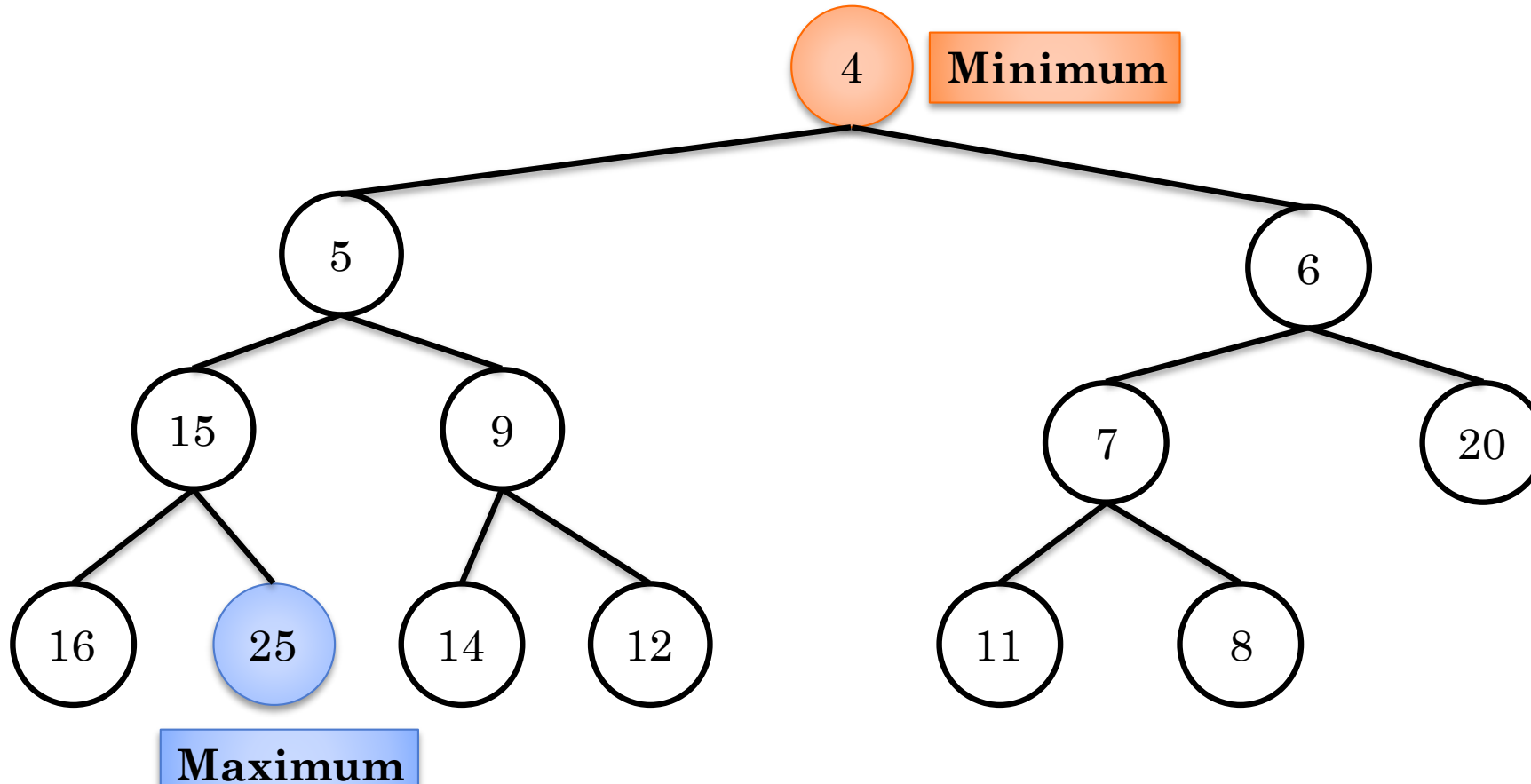
III.TAS: DÉFINITION

- ❖ Un TAS (HEAP) inventé par Williams Floyd en 1964, est un arbre binaire qui vérifie les deux propriétés suivantes :
 - **Propriété structurelle:** arbre binaire parfait, i.e. Tous les niveaux sont totalement remplis sauf le dernier qui est rempli de la gauche vers la droite.
 - **Propriété d'ordre :**
 - TAS_{min} : Clé (père) $\leq$ Clé(fils)
 - TAS_{max} : Clé (père) $\geq$ Clé (fils)

III.TAS: DÉFINITION

❖ Exemple d'un $TAS_{\min}$

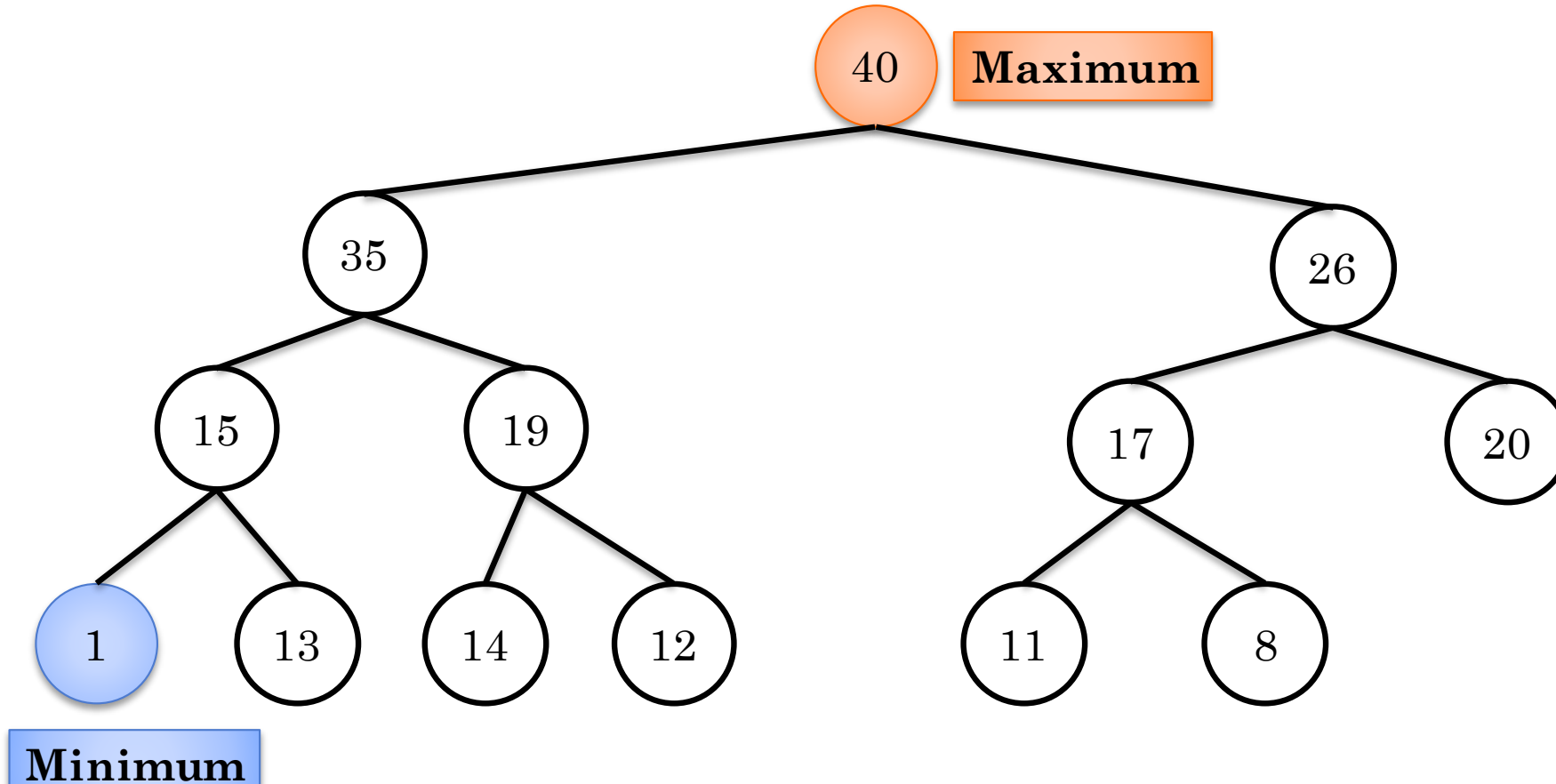
- Clé (père) $\leq$ Clé (fils)
- Le minimum se trouve toujours à la racine



III.TAS: DÉFINITION

❖ Exemple d'un $TAS_{\max}$

- Clé (père) $\geq$ Clé (fils)
- Le maximum se trouve toujours à la racine

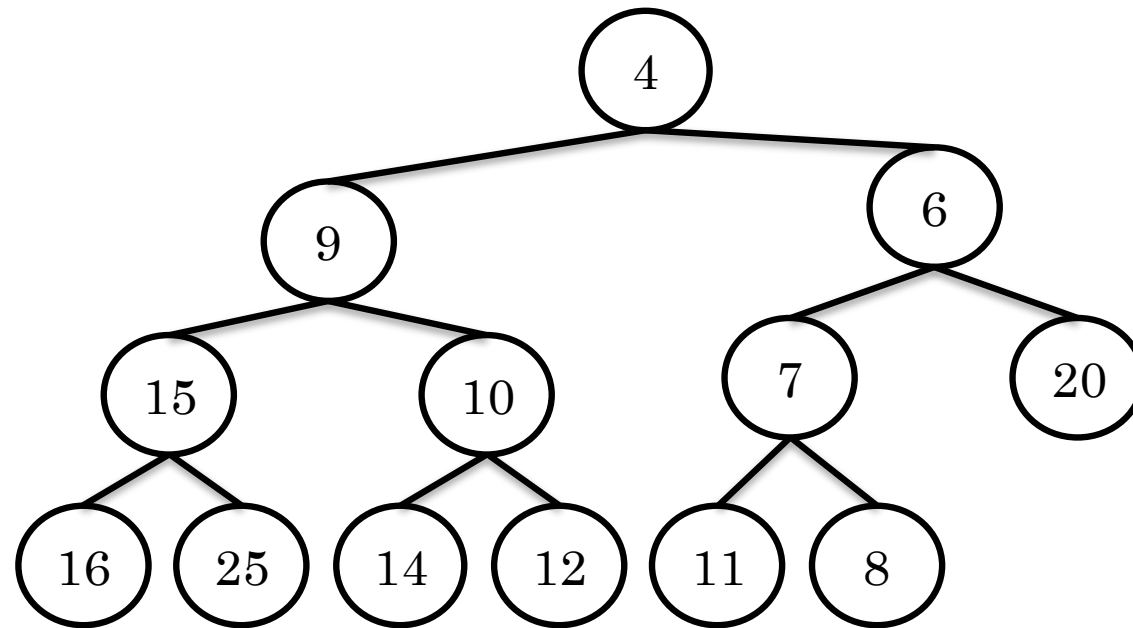


III.TAS: INSERTION

- ❖ Pour insérer une valeur « v » dans un $TAS_{\min}$ [ou $TAS_{\max}$]
 1. Insérer la valeur « v » à la fin du dernier niveau de l'arbre.
 2. Tant que la valeur du père de « v » est plus grande [**petite**] que « v », échanger la valeur du père de v avec « v ».
- ❖ L'étape 1 permet de vérifier la propriété structurelle et l'étape 2 permet de vérifier la propriété de l'ordre.

III.TAS: INSERTION

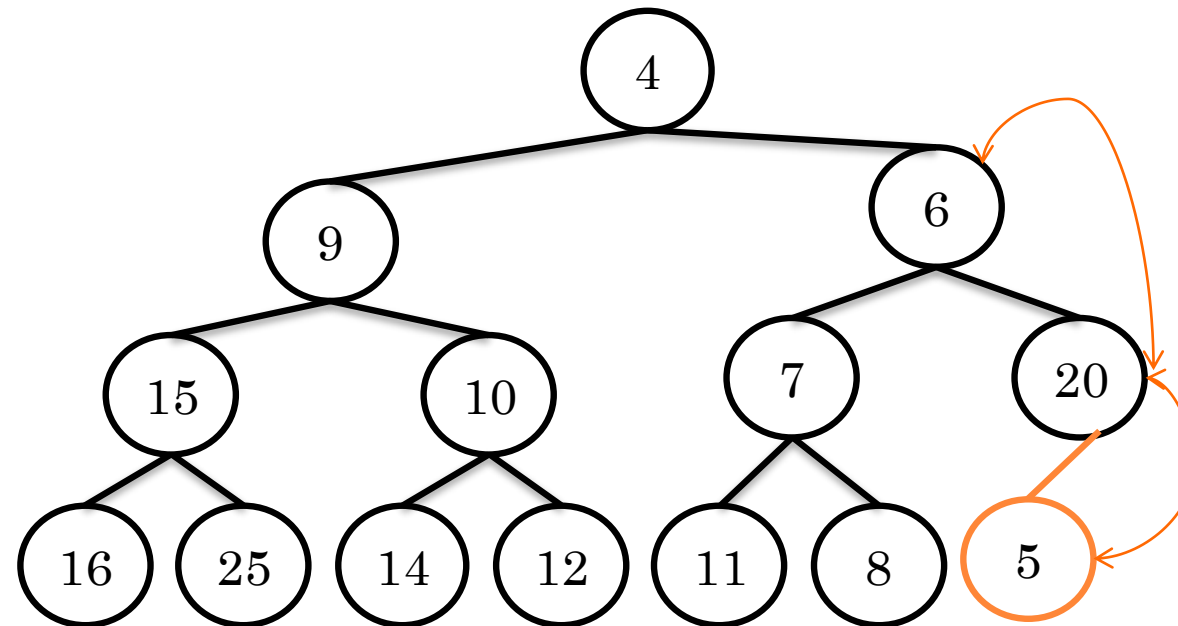
- ❖ **Exemple:** Soit le $TAS_{\min}$ suivant. Donner le résultat après l'insertion 5, 17, 3, 18



III.TAS: INSERTION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

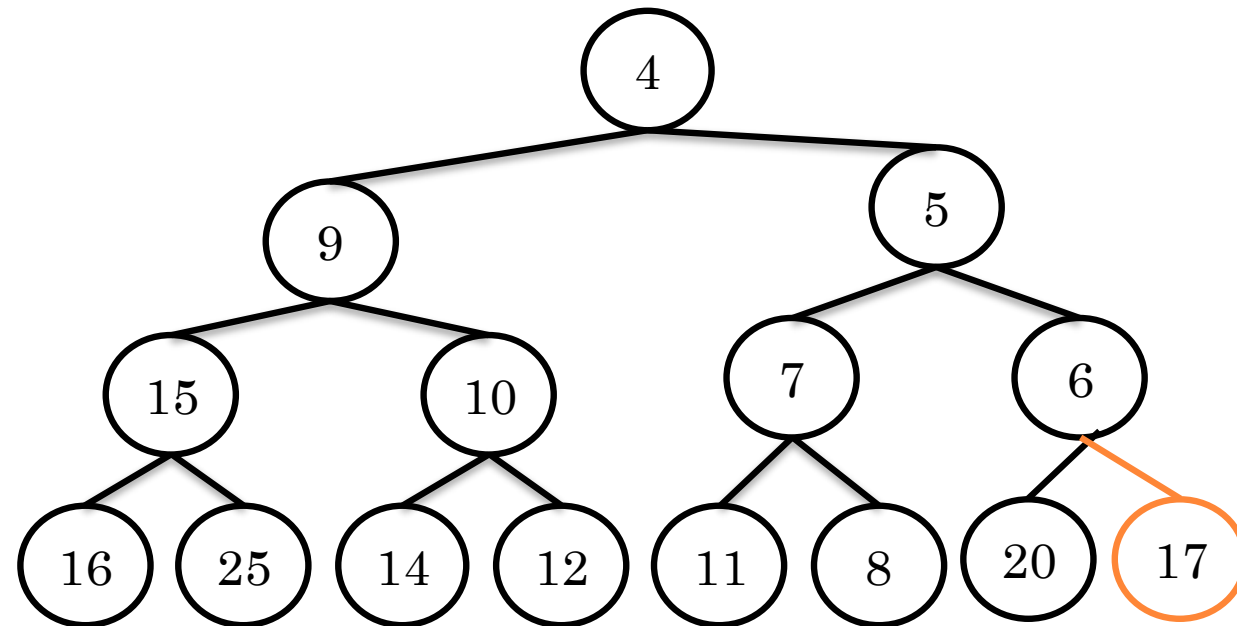
➤ Insertion 5



III.TAS: INSERTION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

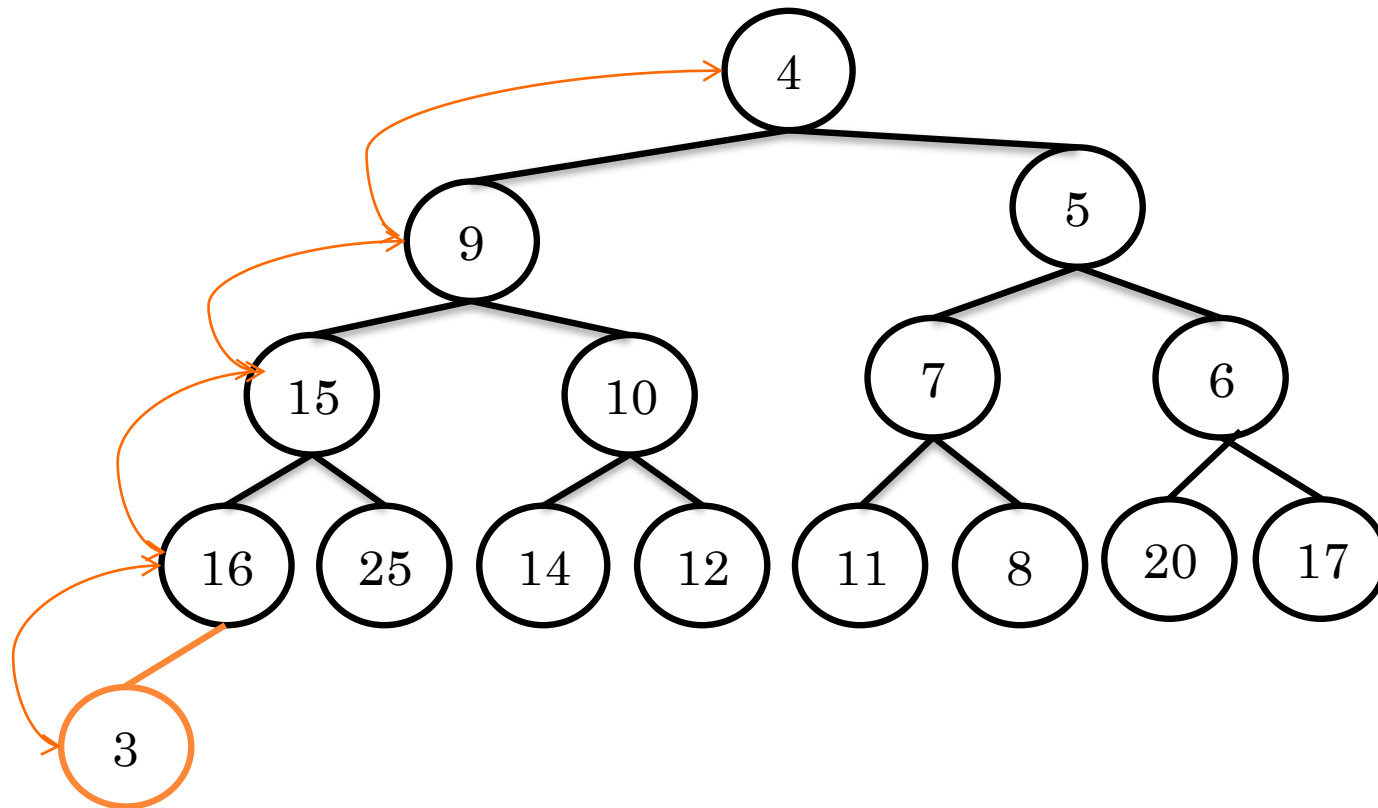
❖ Insertion 17



III.TAS: INSERTION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

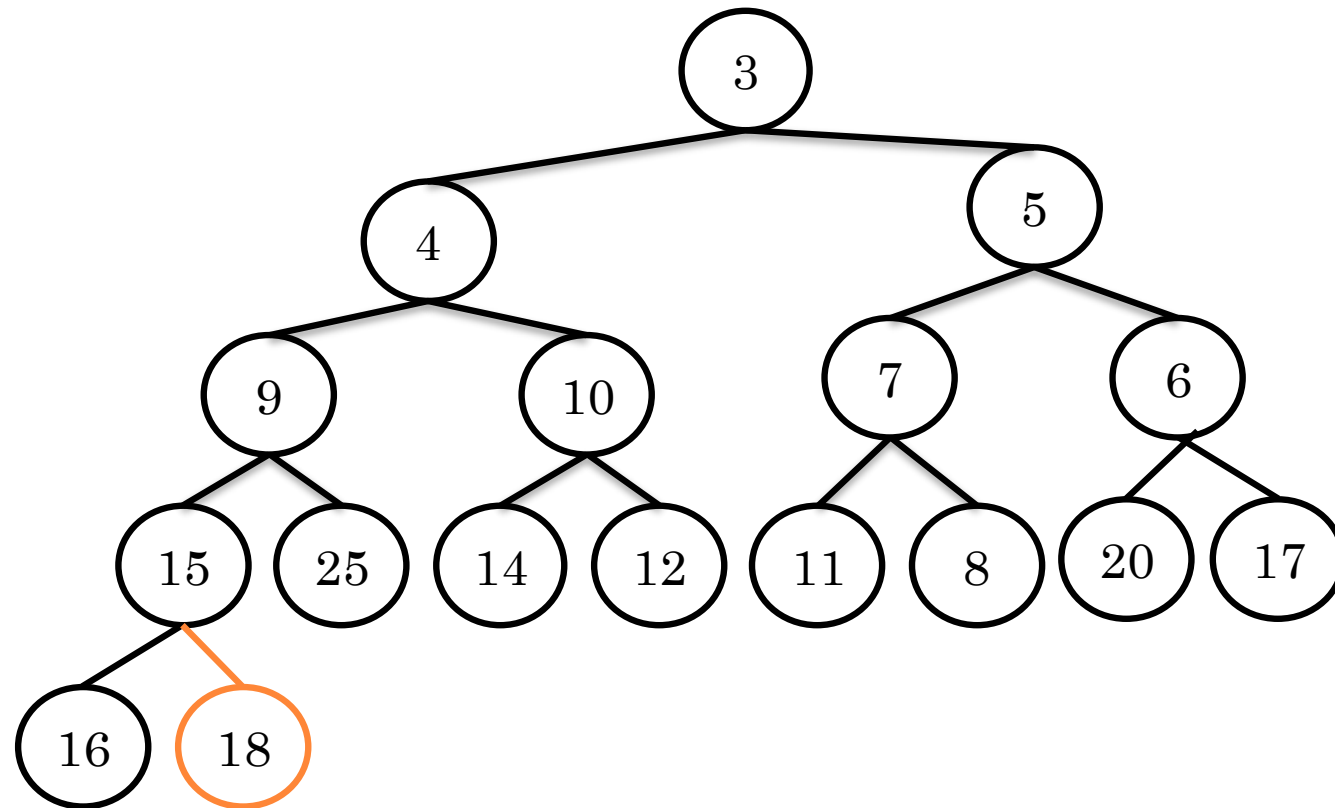
❖ Insertion 3



III.TAS: INSERTION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

❖ Insertion 18

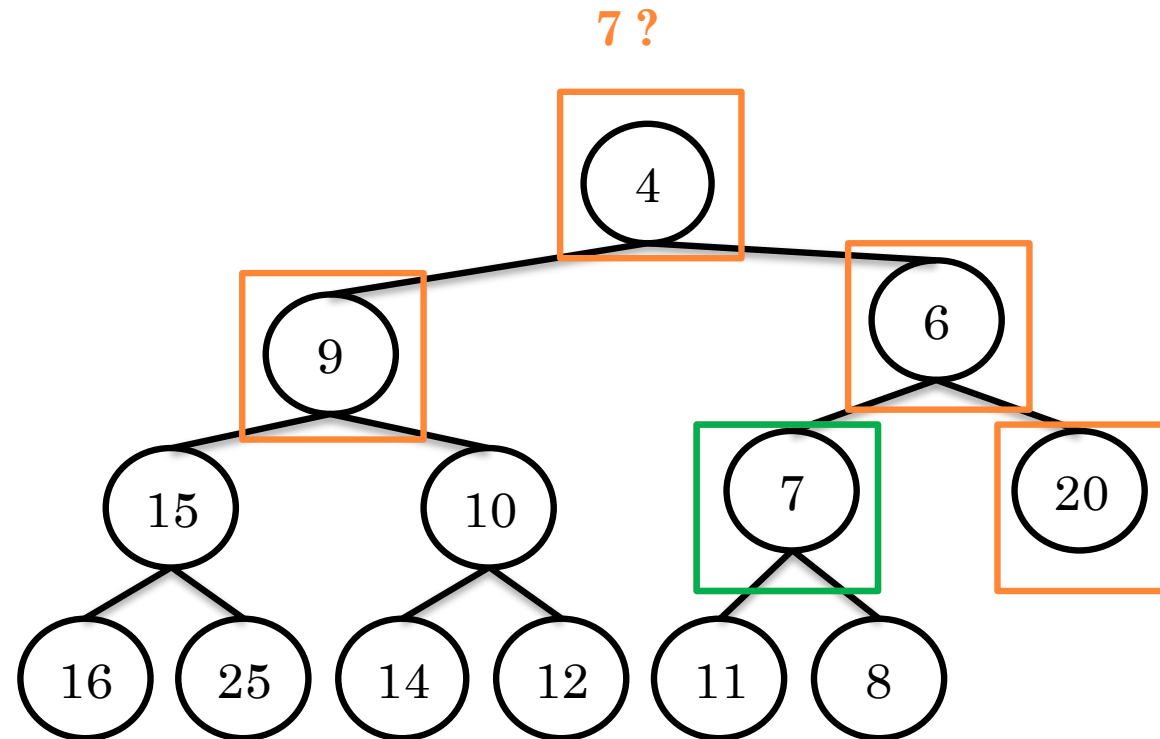


III.TAS: RECHERCHE

- ❖ Pour rechercher une valeur « v » dans un $TAS_{\min}$ [ou $TAS_{\max}$], on doit parcourir l'arbre en largeur (niveau par niveau)
 - On passera au niveau « i » si seulement si la valeur « v » est supérieure [inférieure] à la valeur de l'un des nœuds de niveau « $i-1$ ».

III.TAS: RECHERCHE

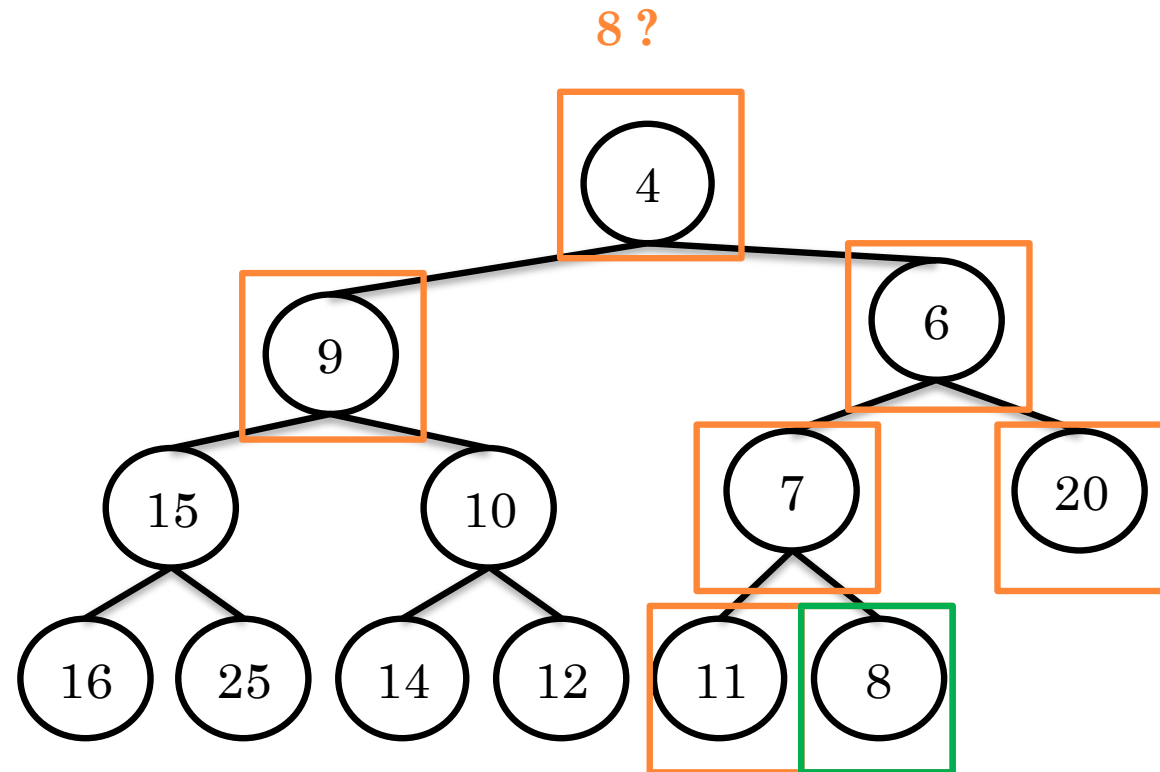
❖ **Exemple:** Soit le $TAS_{\min}$ suivant.



La valeur existe, fin de recherche

III.TAS: RECHERCHE

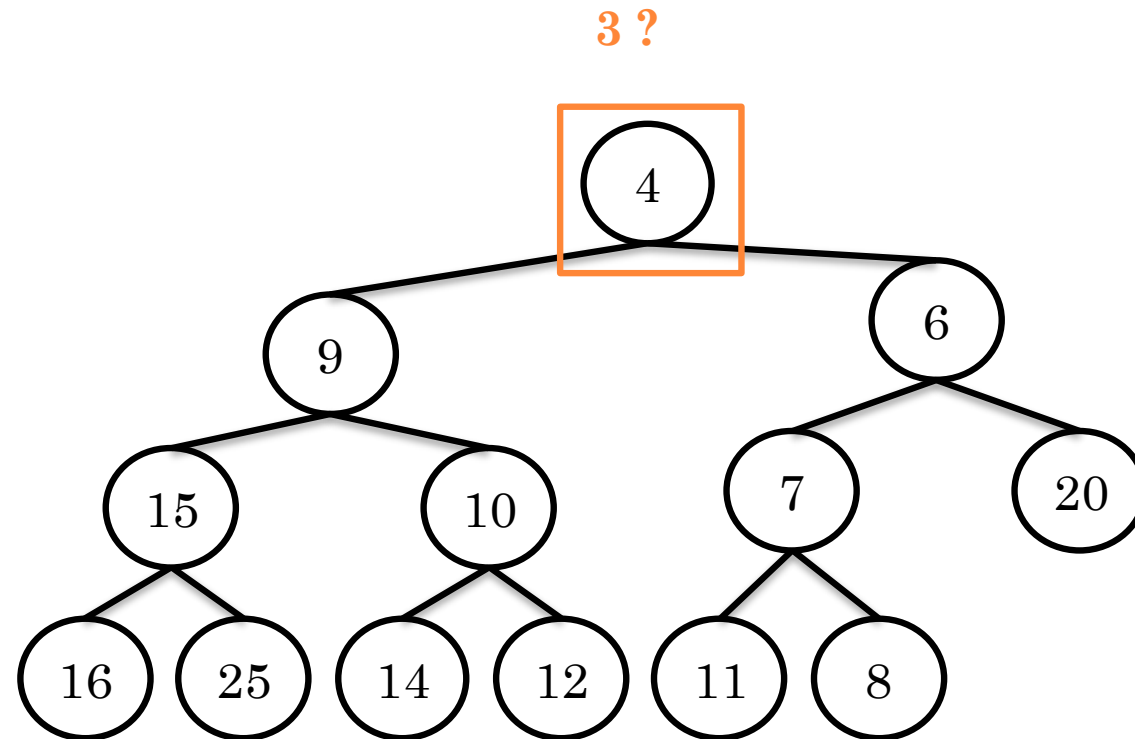
❖ **Exemple:** Soit le $TAS_{\min}$ suivant.



La valeur existe, fin de recherche

III.TAS: RECHERCHE

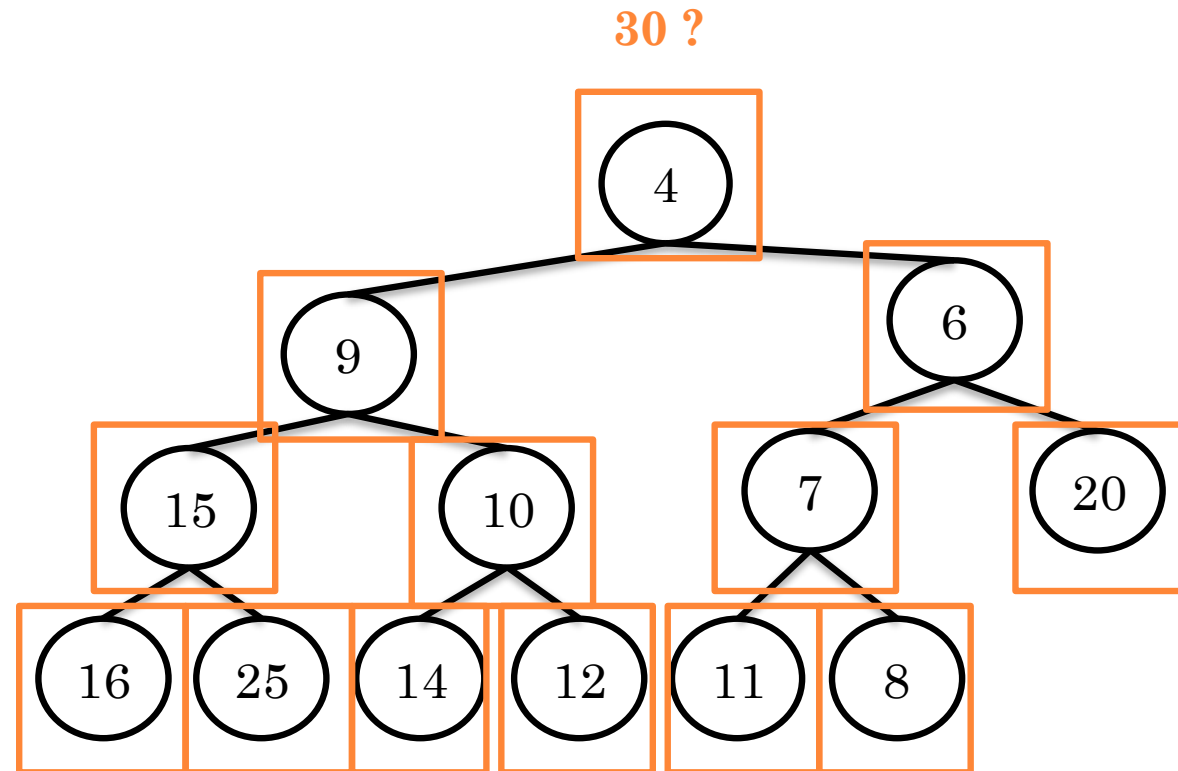
❖ **Exemple:** Soit le $TAS_{\min}$ suivant.



La valeur n'existe pas , fin de recherche

III.TAS: RECHERCHE

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.



Fin de recherche, la valeur n'existe pas

III.TAS: SUPPRESSION

- ❖ Pour supprimer une valeur « v » dans un $TAS_{\min}$ [ou $TAS_{\max}$]
 1. Rechercher la valeur « v », si elle existe on passe à l'étape suivante, sinon stop.
 2. Soit « P » le nœud contenant la valeur « v », remplacer la valeur de « P » par la valeur du dernier nœud du dernier niveau (soit « Q » ce nœud et « x » sa valeur). Cela permet de vérifier la propriété structurelle.

III.TAS: SUPPRESSION

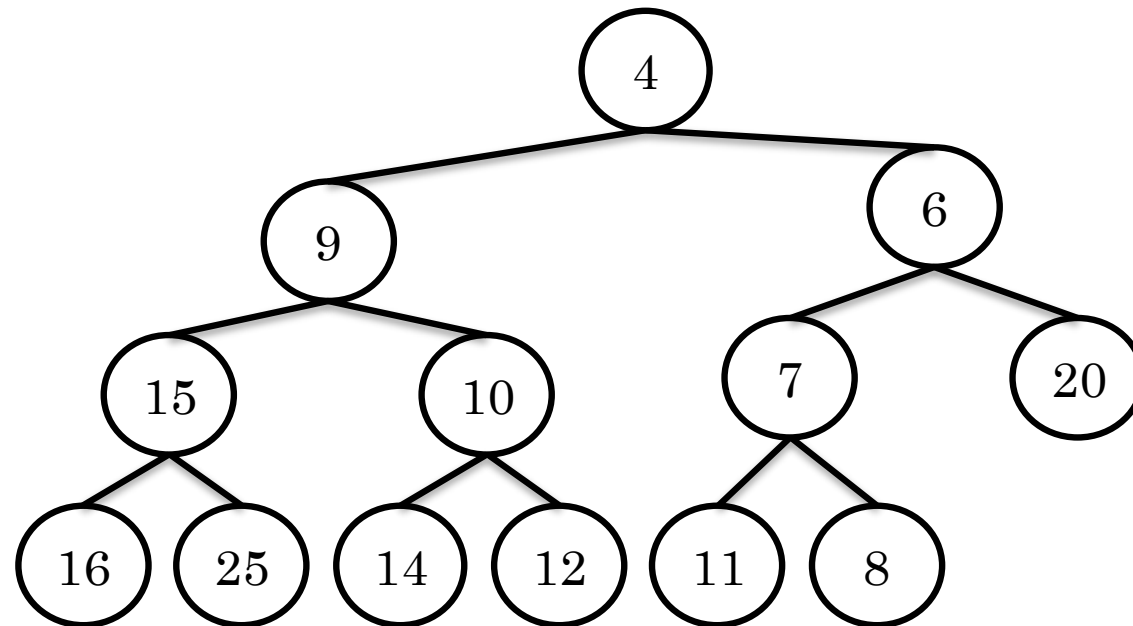
❖ Pour supprimer une valeur « v » dans un $TAS_{\min}$ [ou $TAS_{\max}$]

3. Vérifier la propriété d'ordre:

- a. Tant que la valeur « x » est inférieure [**supérieure**] à celle du père,
échanger la valeur « x » avec celle du père.
- b. Tant que la valeur « x » est supérieure [**inférieure**] à celle de l'un de
ses fils, échanger la valeur « x » avec celle du plus petit [**grand**] de ses
fils.

III.TAS: SUPPRESSION

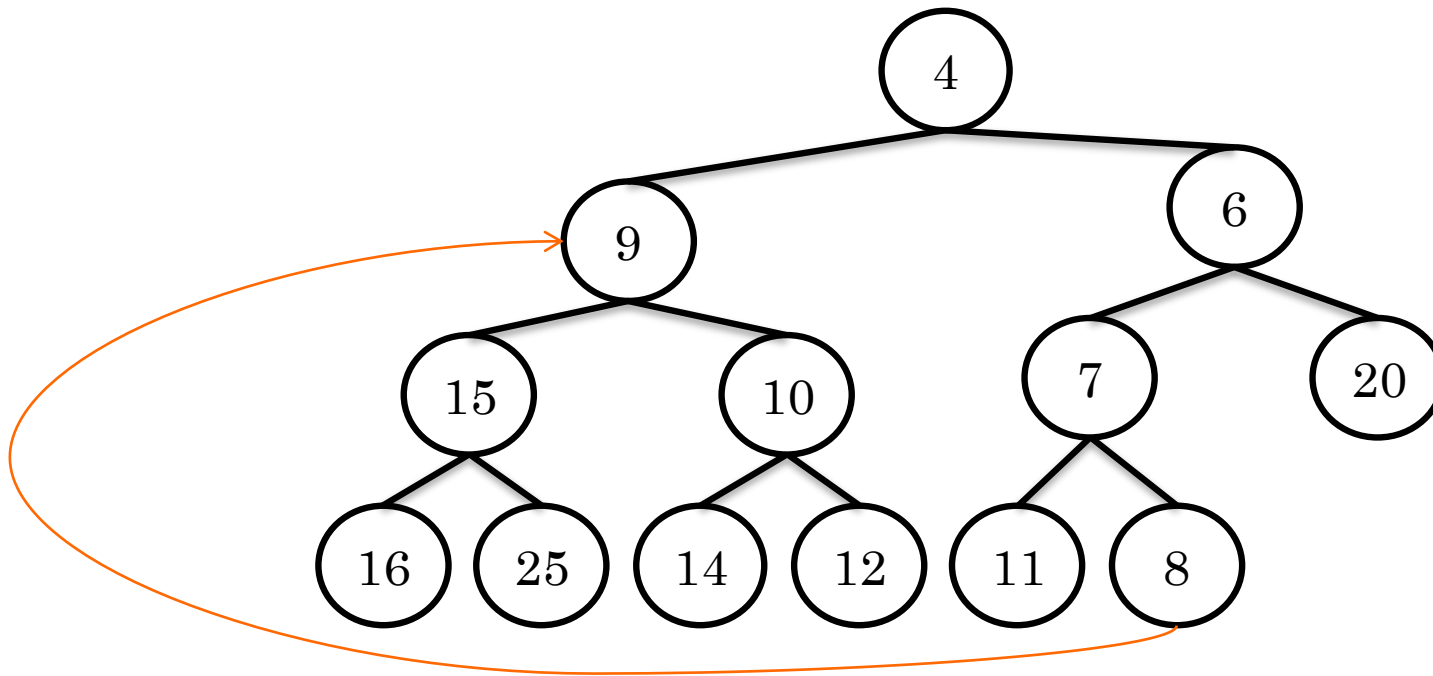
- ❖ **Exemple:** Soit le TAS_{min} suivant. Donner le résultat après la suppression de 9, 16, 6 et 4.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

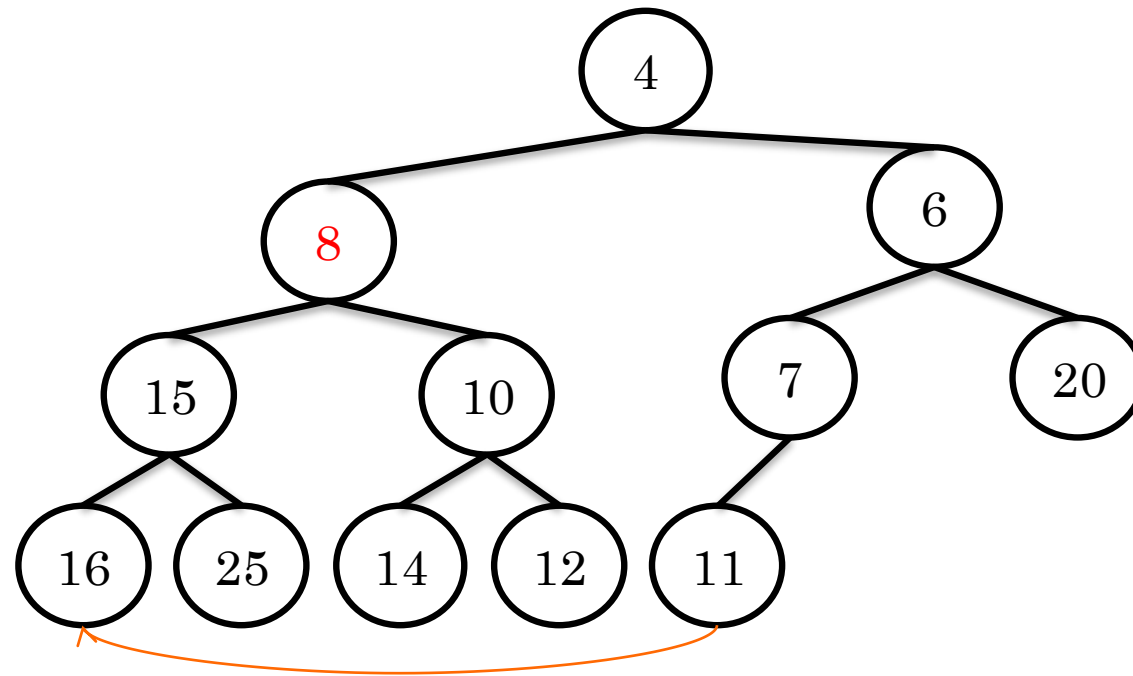
➤ Suppression de 9.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

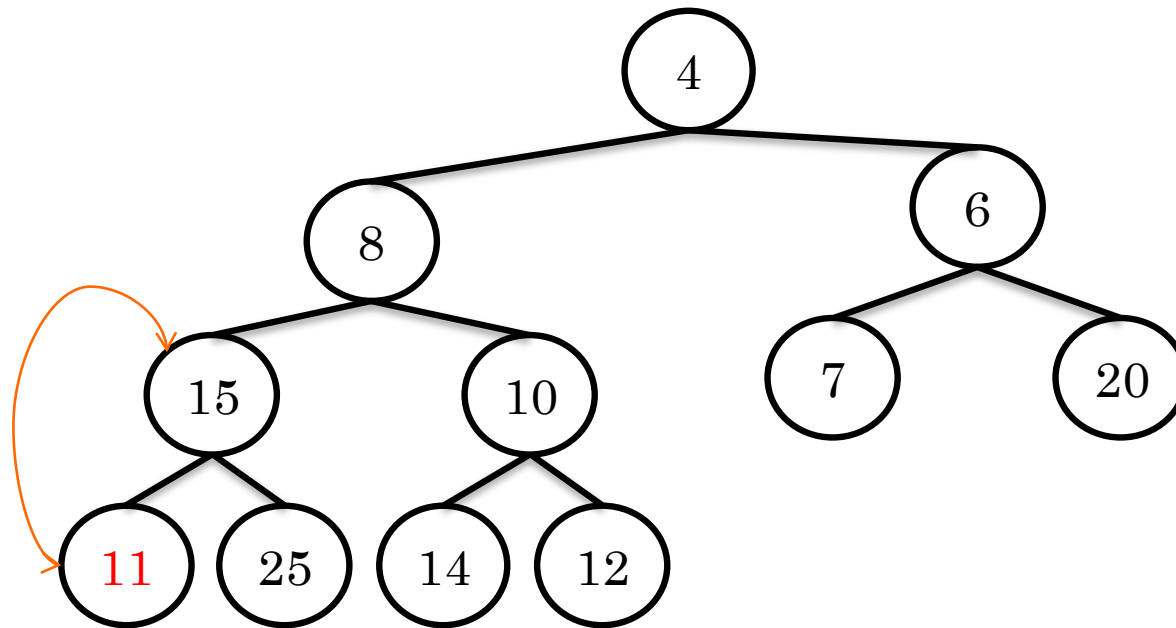
➤ Suppression de 9, 16.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

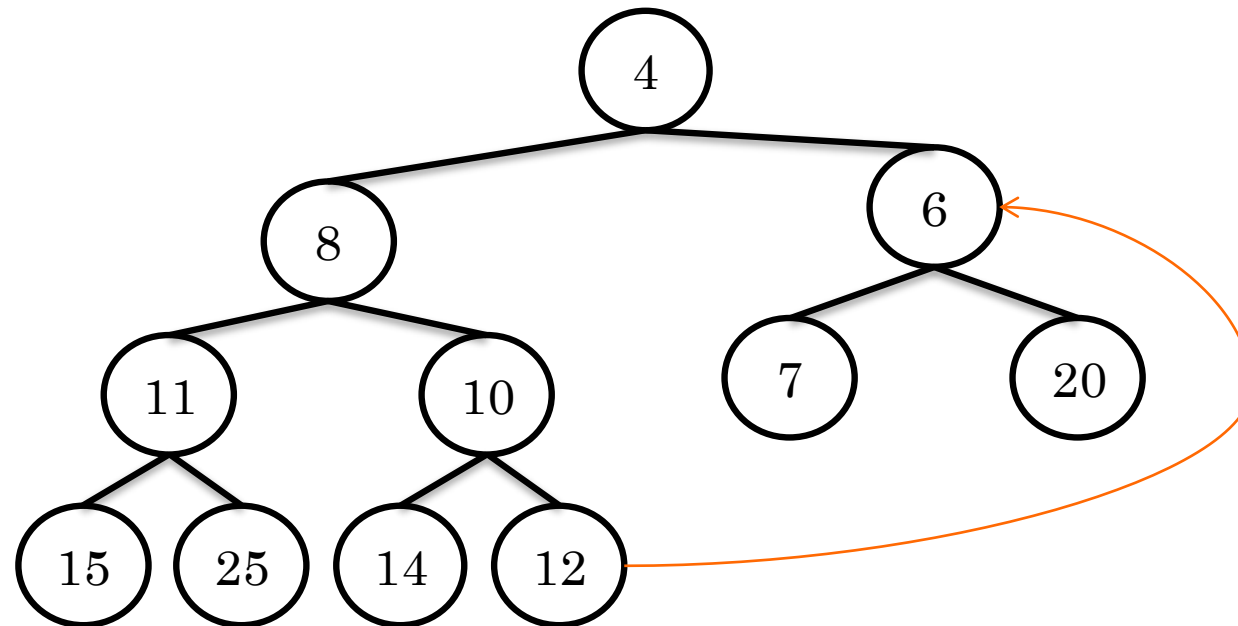
➤ Suppression de 16.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

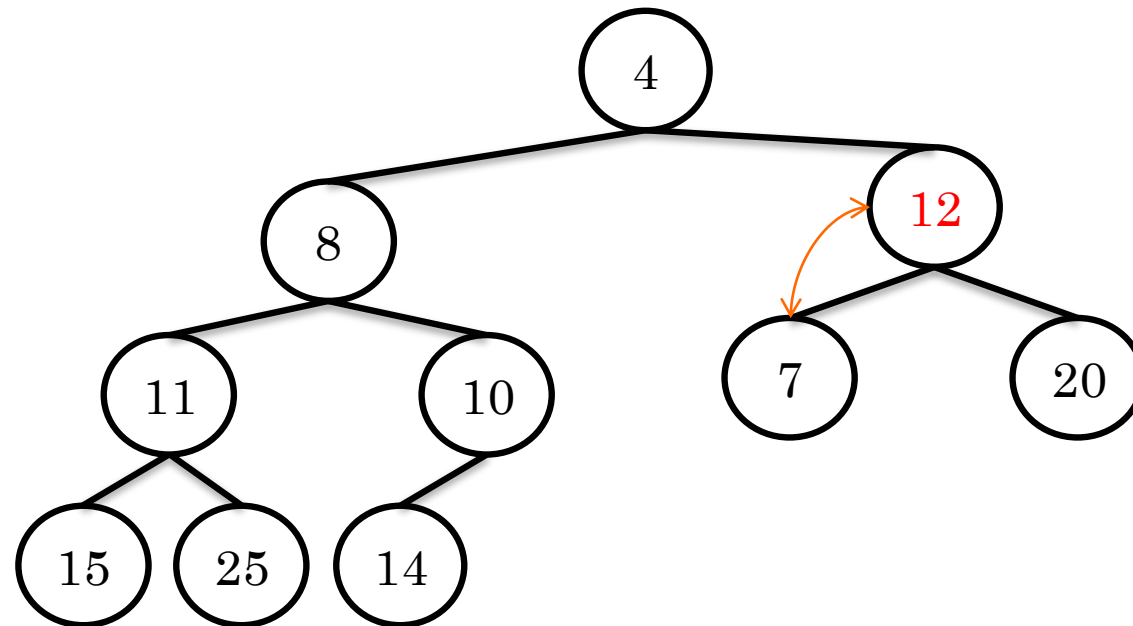
➤ Suppression de 6.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

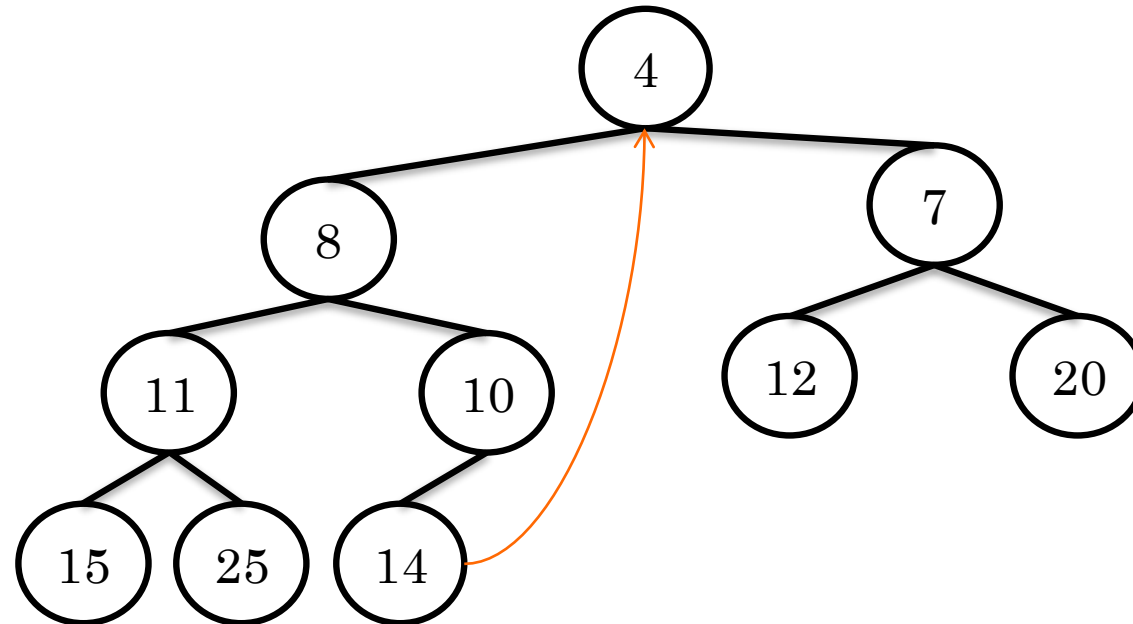
➤ Suppression de 6.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

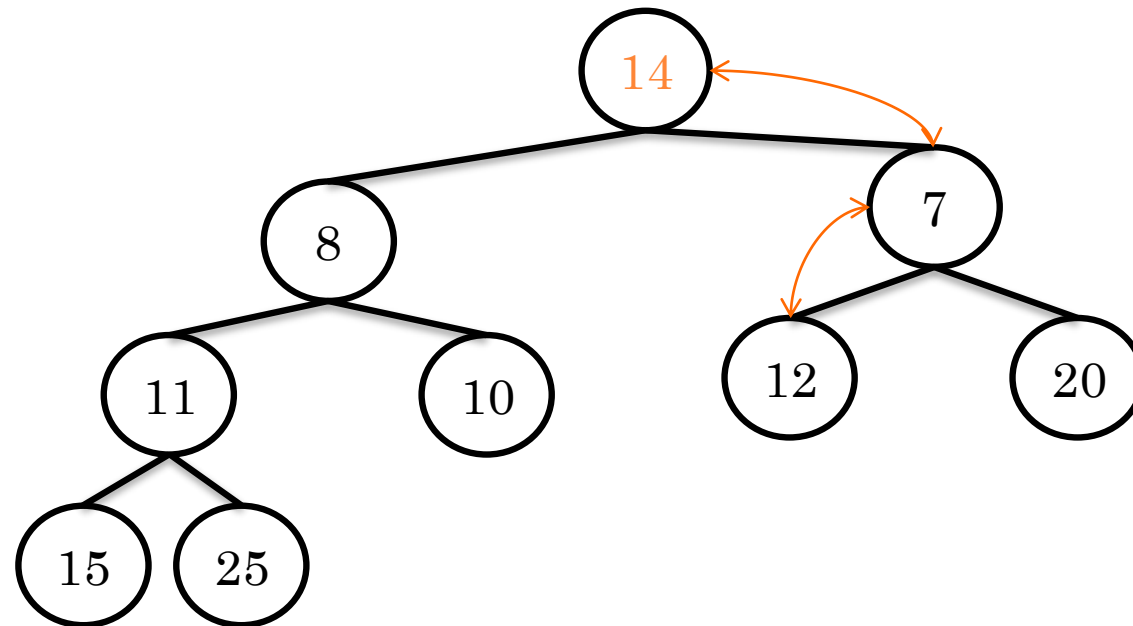
➤ Suppression de 4.



III.TAS: SUPPRESSION

❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

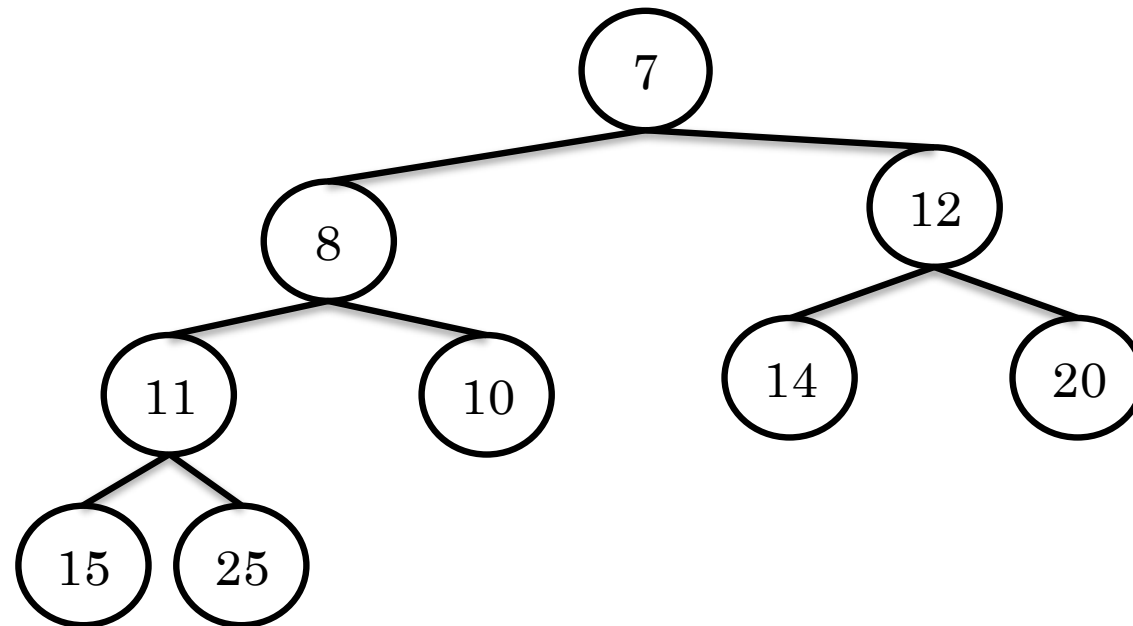
➤ Suppression de 4.



III.TAS: SUPPRESSION

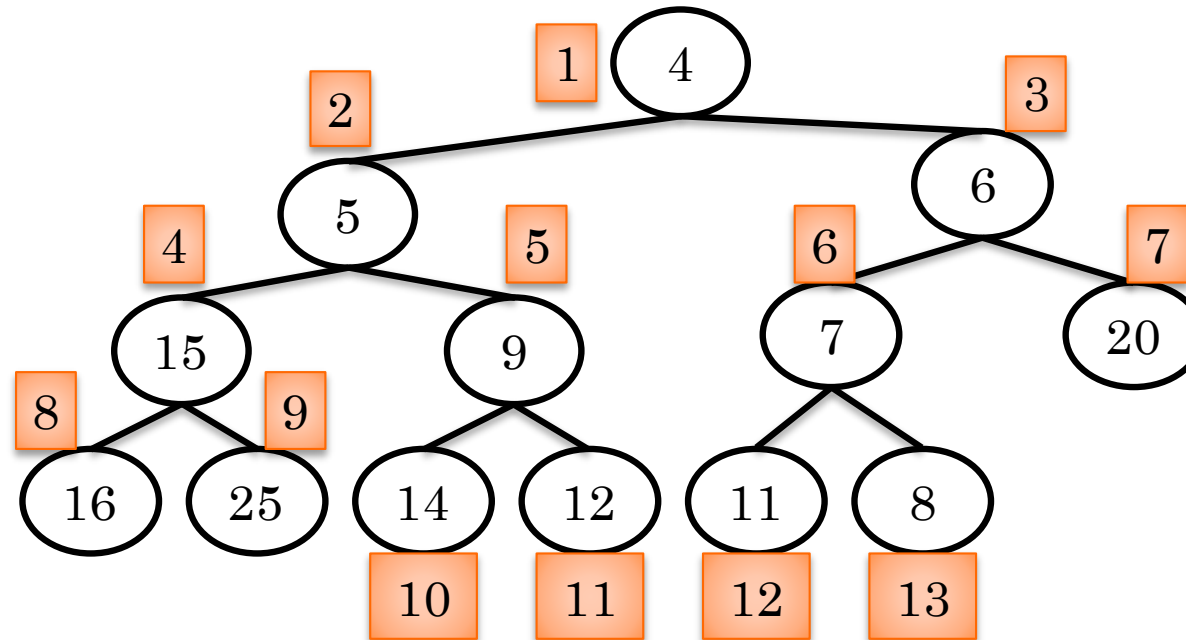
❖ **Exemple:** Soit le $TAS_{\min}$ suivant.

➤ Suppression de 4.



III.TAS: IMPLÉMENTATION

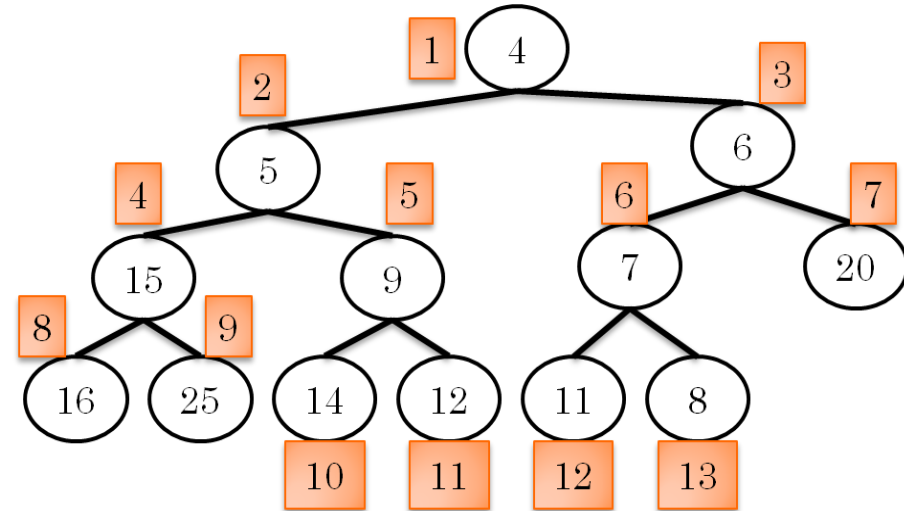
- ❖ Un TAS se représente naturellement par un tableau:
 - Les sommets sont numérotés par un parcours en largeur, de gauche à droite.
 - Le sommet « i » est rangé dans la case d'indice i du tableau.



III.TAS: IMPLÉMENTATION

❖ On parle ici de la **représentation statique séquentielle** d'un arbre binaire

- La case 0 est vide
- $\text{Indice}(\text{racine})=1$
- $\text{Indice}(\text{FG})=2*\text{Indice}(\text{Père})$
- $\text{Indice}(\text{FD})=2*\text{Indice}(\text{Père})+1$
- $\text{Indice}(\text{Père})= \lfloor \text{Indice}(\text{Fils})/2 \rfloor$



0	1	2	3	4	5	6	7	8	9	10	11	12	13
??	4	5	6	15	9	7	20	16	25	14	12	11	8

PARTIE IV:

ARBRES M-AIRE DE

RECHERCHE (AMR)

PLAN DE LA PARTIE IV

- ❑ Définition

- ❑ Opérations de Base: Recherche, Insertion et
Suppression

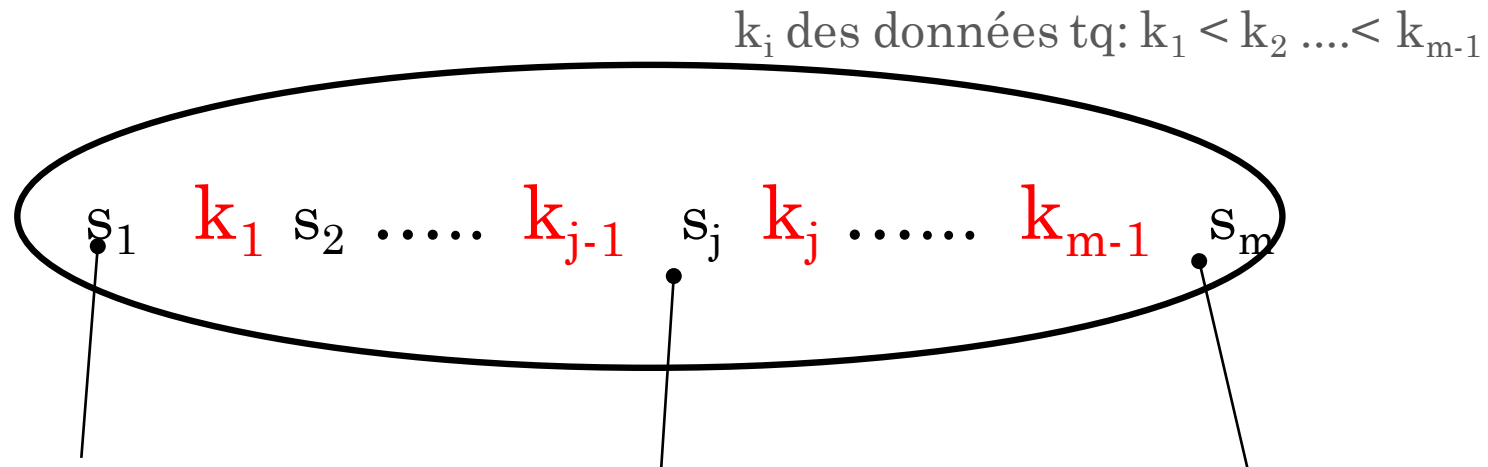
IV. DÉFINITION

- ❖ Un Arbre M-aire de Recherche (AMR) peut être défini comme une généralisation de l'arbre binaire de recherche. Au lieu d'avoir une clé et deux pointeurs, on aura « $m-1$ » clés et « m » pointeurs.
- ❖ Un AMR d'ordre « m » est un arbre dans lequel chaque nœud peut avoir « m » fils.

IV. DÉFINITION

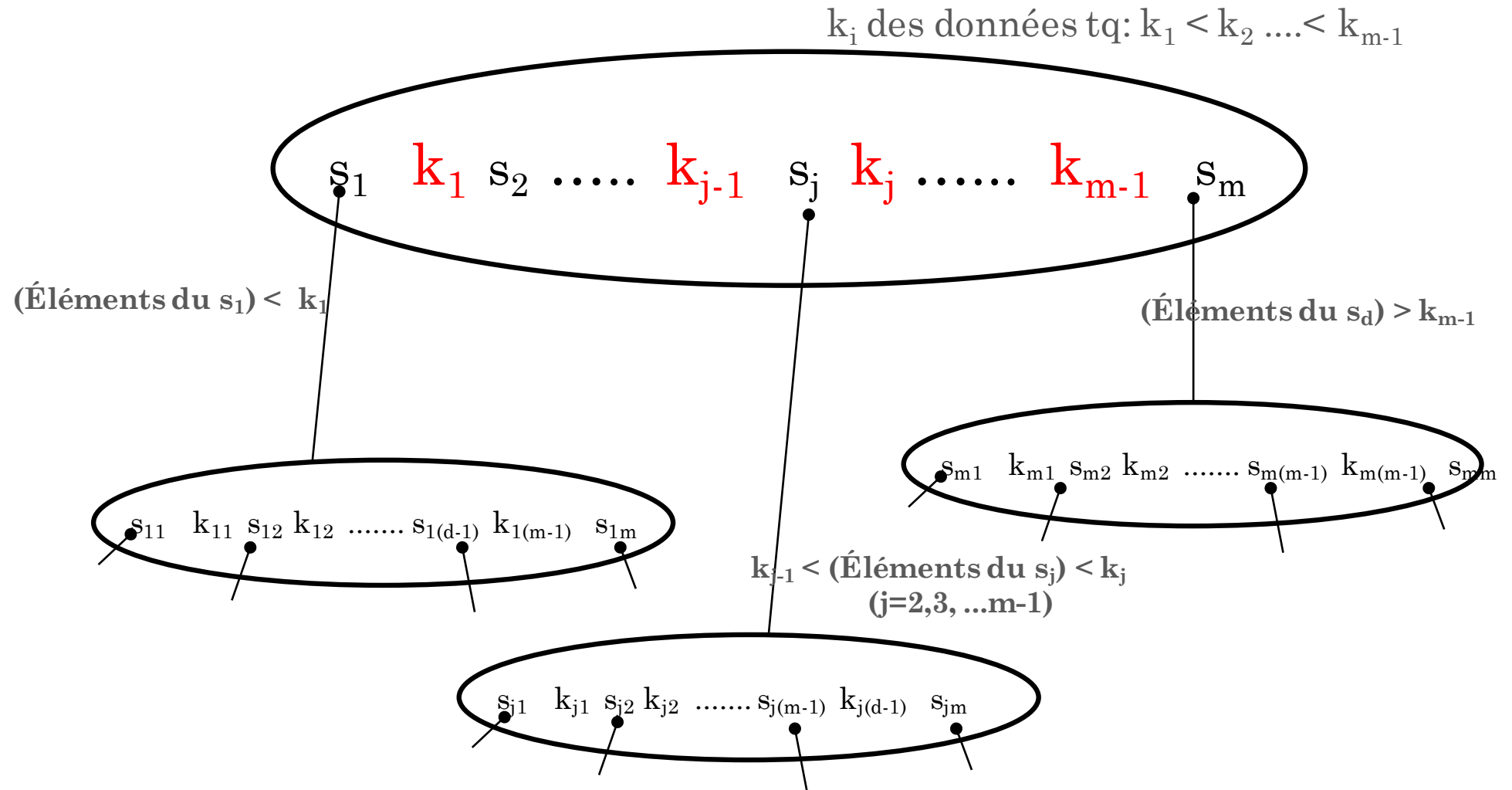
❖ Si $s_1, s_2, \dots, s_m$ sont les « m » sous arbres issus d'un nœud donné avec les clés $k_1, k_2, \dots, k_{m-1}$ dans l'ordre ascendant, alors :

- Toutes les clés dans s_1 sont inférieures à k_1
- Toutes les clés dans s_j ($j=2,3, \dots, m-1$) sont supérieures à k_{j-1} et inférieures à k_j
- Toutes les clés dans s_m sont supérieures k_{m-1} .



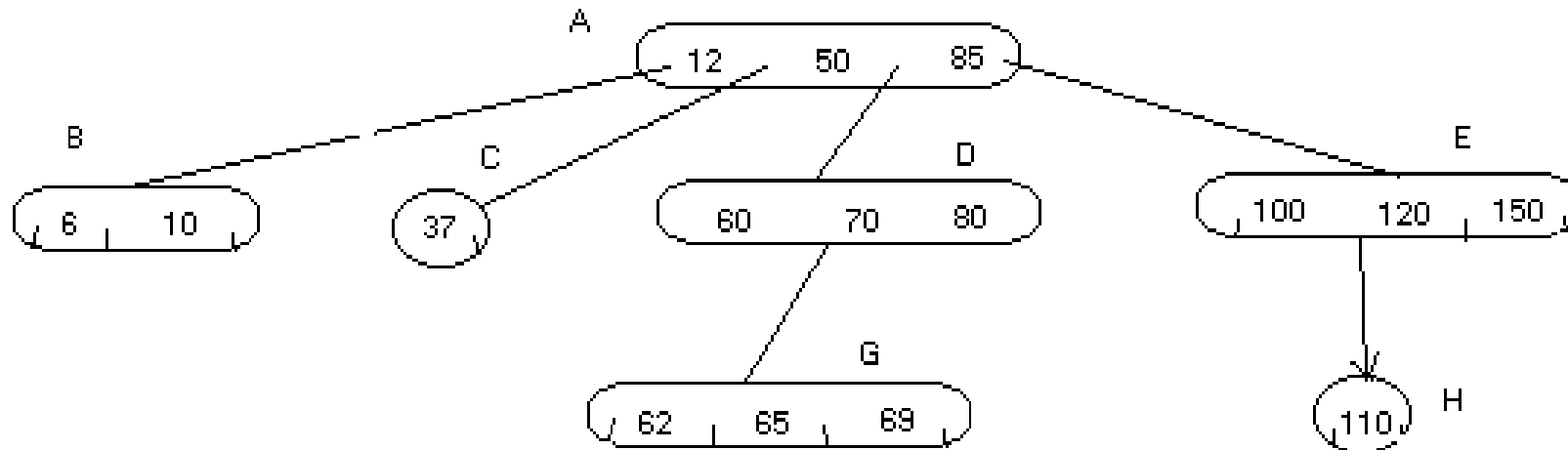
IV. DÉFINITION

❖ Propriété d'ordre:



IV. DÉFINITION

❖ **Exemple:** Soit l'AMR d'ordre 4 suivant

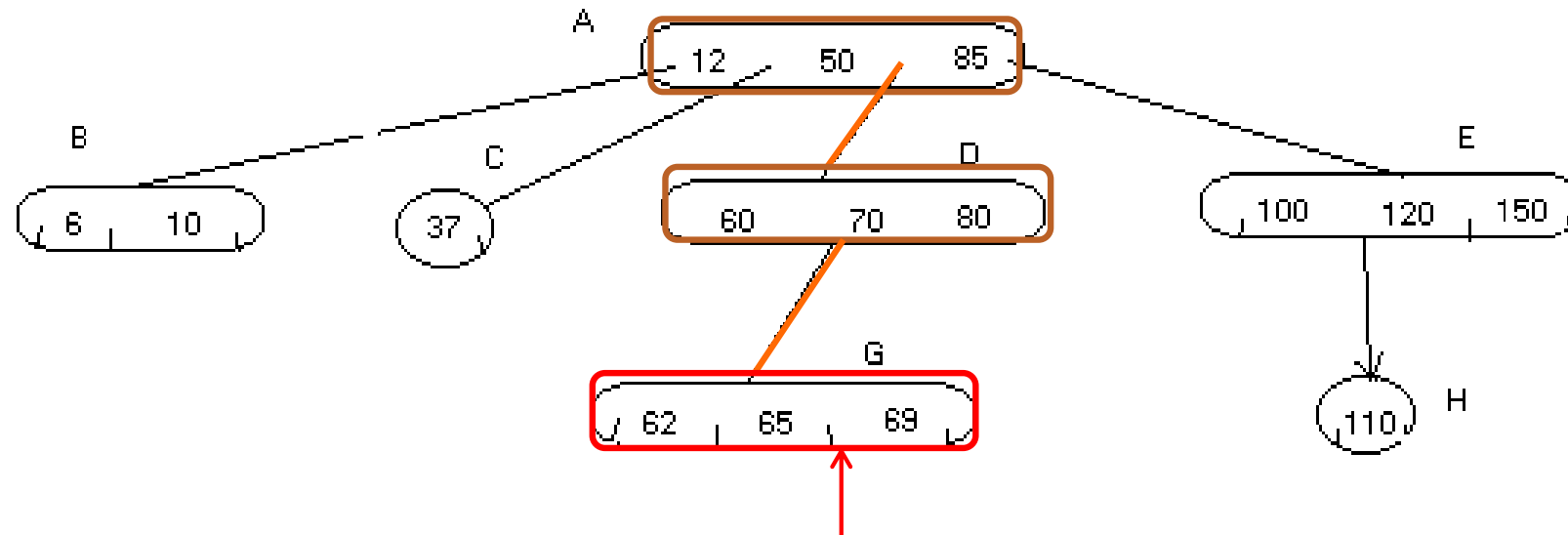


IV. RECHERCHE

❖ La recherche dans un AMR ressemble beaucoup à celle effectuée dans un ABR, excepté qu'au lieu de prendre à chaque nœud une décision de branchement binaire (Fils gauche ou droit), on prend une décision à options multiples, selon le nombre de fils du nœud.

IV. RECHERCHE

❖ Exemple: Rechercher l'élément 68.



Nil i.e: L'élément n'existe pas

IV. INSERTION

- ❖ **L'insertion se fait toujours au niveau des feuilles.**
- ❖ L'insertion se déroule comme suit:
 1. Rechercher la position de la clé à insérer « x », soit « P » le nœud trouvé et « Q » son père.
 2. Si « P » est à Nil alors créer un nouveau nœud contenant la clé « x » ensuite le chaîner à la bonne position comme fils du « Q ».

Sinon /* « P » est une feuille*/

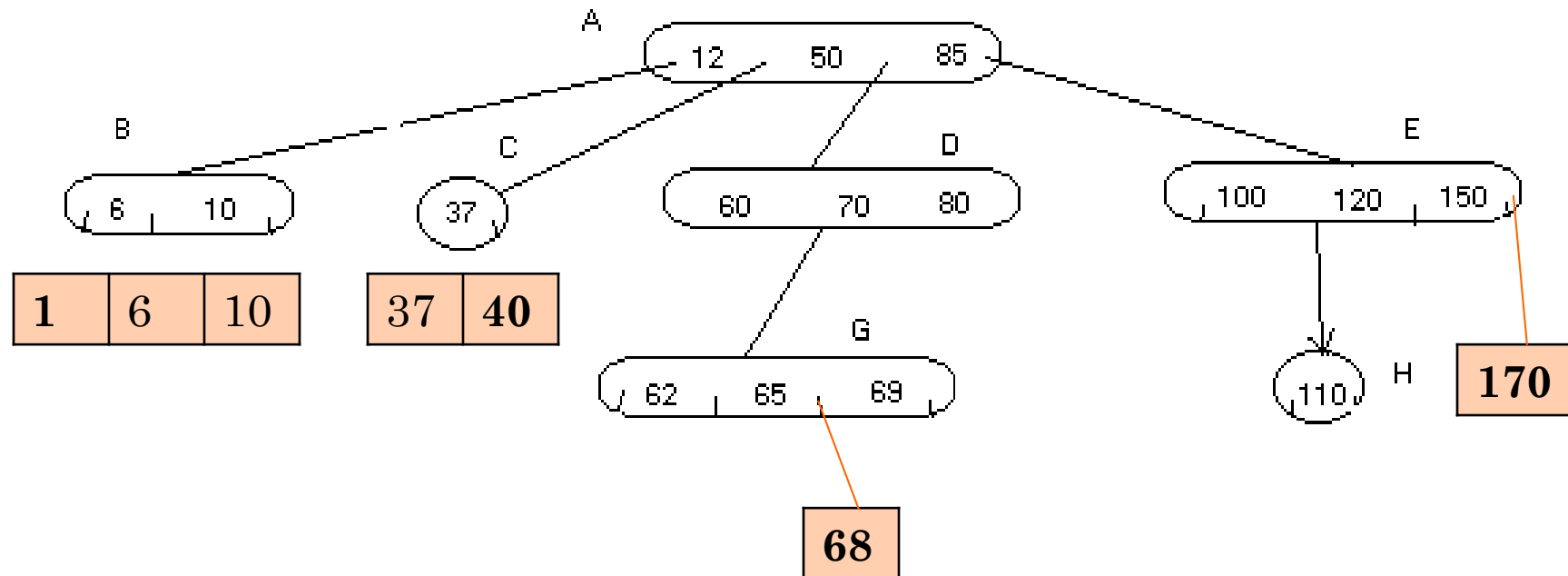
 - Si « P » n'est pas pleine (le nombre des clés insérées est strictement inférieur à $m-1$) alors insérer la clé à sa bonne position dans la feuille.

IV. INSERTION

- ❖ **L'insertion se fait toujours au niveau des feuilles.**
- ❖ L'insertion se déroule comme suit:
 2. Sinon /* « ***P*** » est une feuille*/
 - Si « ***P*** » n'est pas pleine alors insérer la clé à sa bonne position dans la feuille.
 - Sinon, la feuille est pleine (le nombre des clés insérés est égale à $m-1$, i.e. il y a un débordement) alors créer un nouveau nœud contenant la clé « ***x*** » ensuite le chaîner à la bonne position comme fils du « ***P*** »

IV. INSERTION

❖ Exemple: Soit R un AMR d'ordre 4. Insérer les éléments suivants dans R: 1, 40, 68, 170.



IV. SUPPRESSION

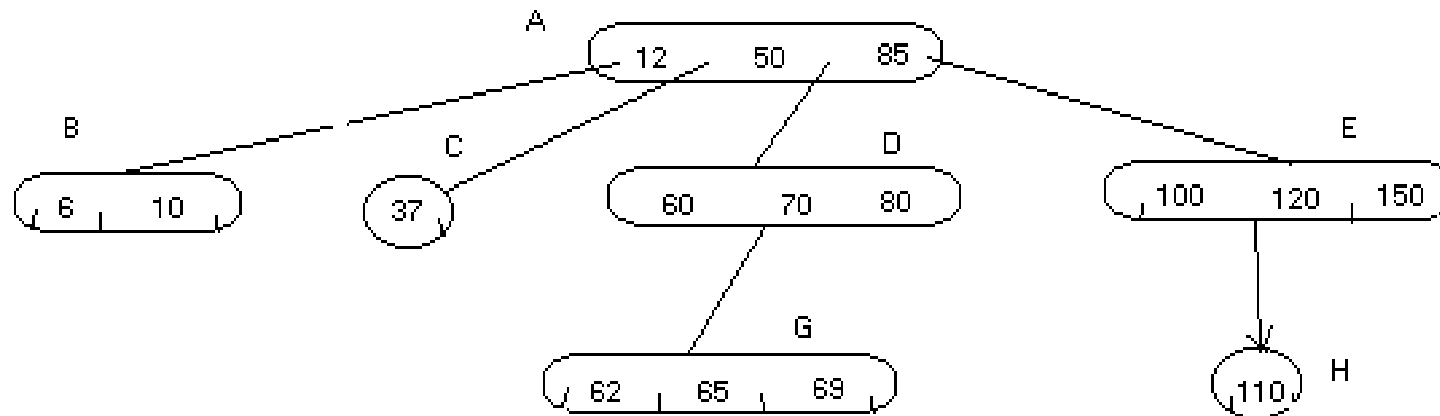
❖ Technique similaire à celle des ABR:

1. Rechercher la clé à supprimer « x ». Si elle existe (soit « P » le nœud trouvé) on passe à l'étape suivante, sinon on s'arrête.
2. On distingue trois cas:
 - a. Si la clé « x » n'a aucun sous arbre (i.e. les deux sous arbres gauche ou droit sont vides) alors supprimer la clé, ensuite tasser (ou organiser) le nœud (décalage ou déplacement des clés et/ou des fils).
 - Si la clé à supprimer est la seule clé dans la feuille « P » alors libérer la feuille.

IV. SUPPRESSION

❖ Technique similaire à celle des ABR:

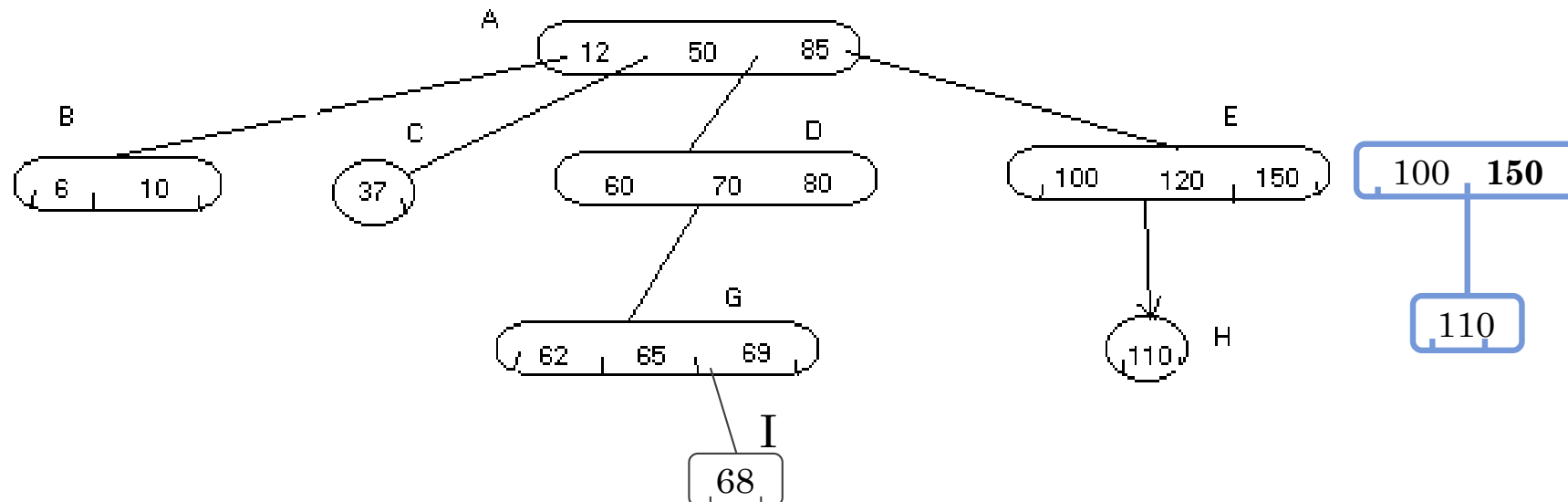
- a. Si la clé « x » n'a aucun sous arbre (i.e. les deux sous arbres gauche ou droit sont vides) alors supprimer la clé, ensuite tasser (ou organiser) le nœud (décalage ou déplacement des clés et/ou des fils).
 - Si la clé à supprimer est la seule clé dans la feuille « *P* » alors libérer la feuille.
 - Exemple: supprimer 10, 37, 110 ou 150,



IV. SUPPRESSION

❖ Technique similaire à celle des ABR:

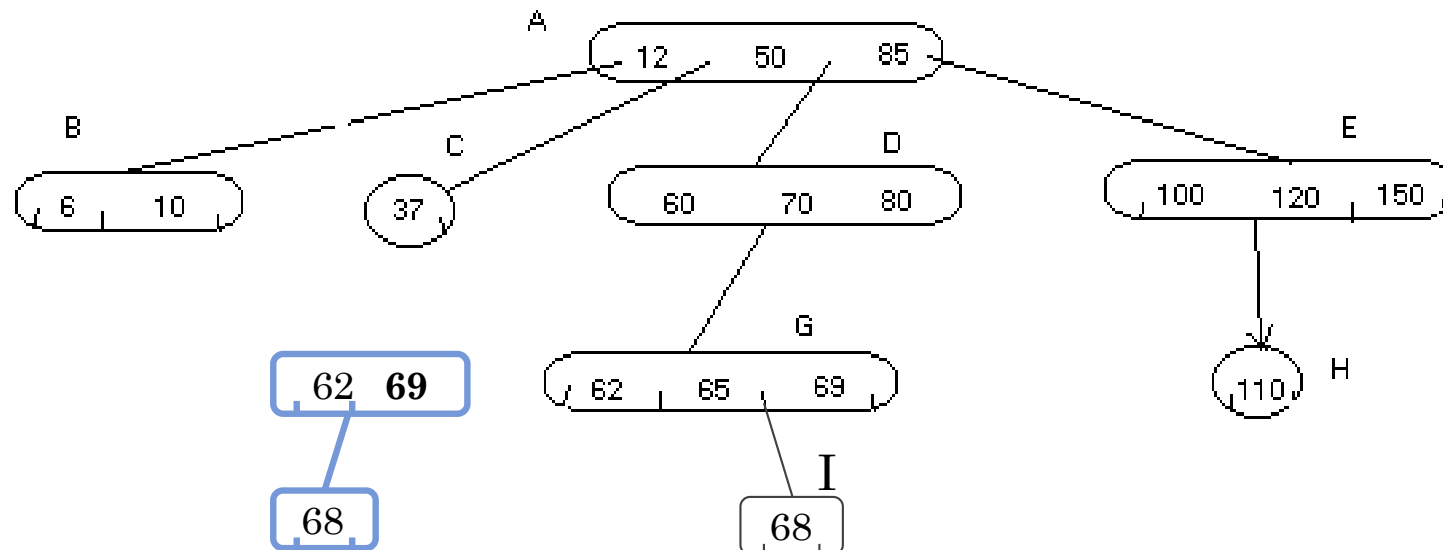
- b. Si la clé « x » a un seul sous arbre vide alors supprimer la clé, ensuite tasser (ou organiser) le nœud (décalage ou déplacement des clés et/ou des fils).
- Exemple: la suppression du 120 entraîne le décalage de 150 à la position de 120.



IV. SUPPRESSION

❖ Technique similaire à celle des ABR:

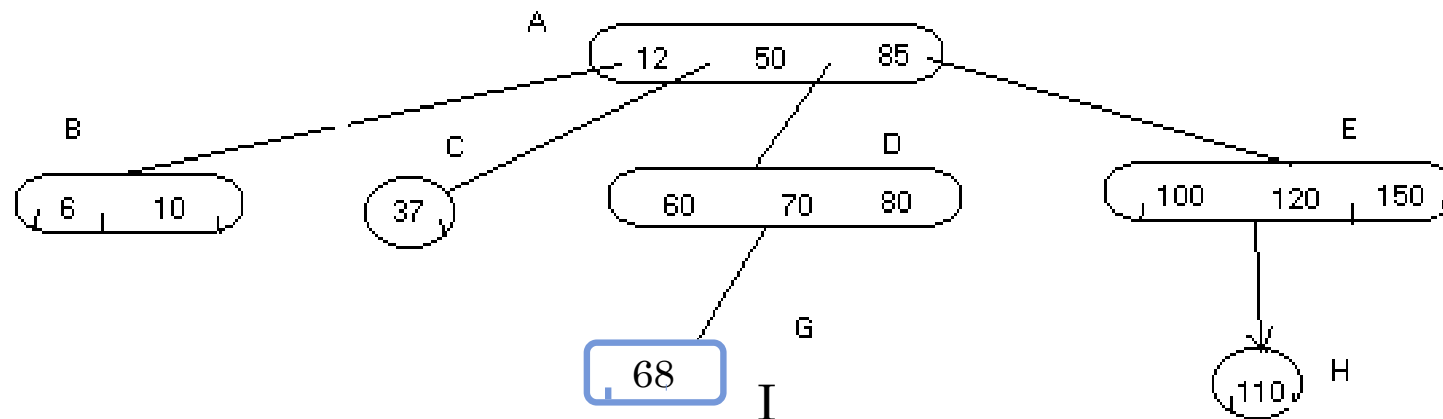
- b. Si la clé « x » a un seul sous arbre vide alors supprimer la clé, ensuite tasser (ou organiser) le nœud (décalage ou déplacement des clés et/ou des fils).
- Exemple: la suppression du 65 entraîne le décalage de 69 à la position de 65 ainsi le déplacement du fils droit de 65.



IV. SUPPRESSION

❖ Technique similaire à celle des ABR:

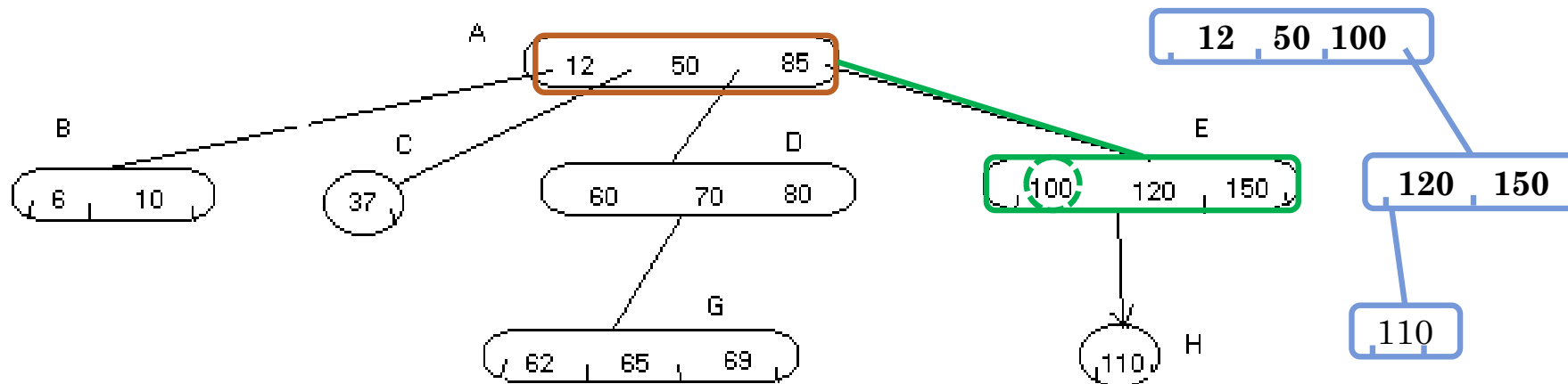
- b. Si l'élément à supprimer a un sous arbre gauche ou droit vide alors supprimer l'élément, ensuite tasser le nœud (décalage des clés et/ou déplacement des fils).
- Si la clé est la seule dans le nœud « *P* » alors chaîner le fils « *P* » du à son père.
- Exemple: la suppression du 62 ensuite 69 entraîne la libération du nœud G et le chaînage du nœud I avec D.

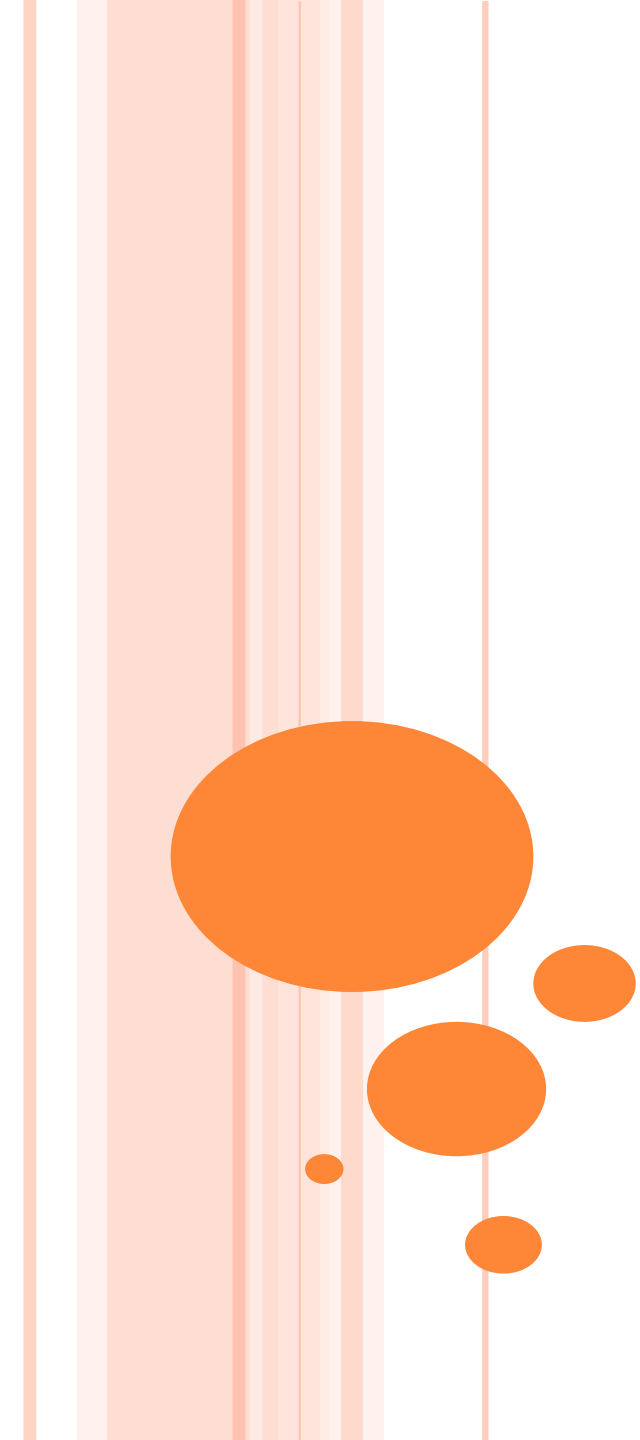


IV. SUPPRESSION

❖ Technique similaire à celle des ABR:

- c. Si la clé « x » a les deux sous arbres non vides alors remplacer cette clé par son successeur/prédécesseur le plus proche (soit la clé « y »), ensuite supprimer la clé « y » (aller au cas a ou b).
- Exemple: la suppression du 85 entraîne le remplacement du 85 par 100, ensuite la suppression du 100





PARTIE V:

B-ARBRES: AMR EQUILIBRÉS

PLAN DE LA PARTIE V

- ❑ Introduction
- ❑ Définition
- ❑ Opérations de Base: Insertion et Suppression

V. INTRODUCTION

- ❖ Les principaux problèmes avec les AMR sont
 - ❖ Équilibrage après MAJ
 - ❖ Gaspilla d'espace dû aux suppression
- ❖ Bayer et McCreight (en 1970) ont fourni une solution à ces problèmes par l'invention des B-arbres (B pour Bayer / Boeng / Balanced).

V. DÉFINITION

❖ Un B-arbre d'ordre m (tel que $m = 2*d + 1$, $d \geq 1$) est défini comme suit :

- La racine, si elle n'est pas une feuille, a au moins 2 fils.
- Chaque nœud contient k clés avec:
 - $1 \leq k \leq 2*d$ (nœud racine)
 - $d \leq k \leq 2*d$ (nœud non racine)
- Tous les nœuds feuilles sont au même niveau.

V. INTRODUCTION

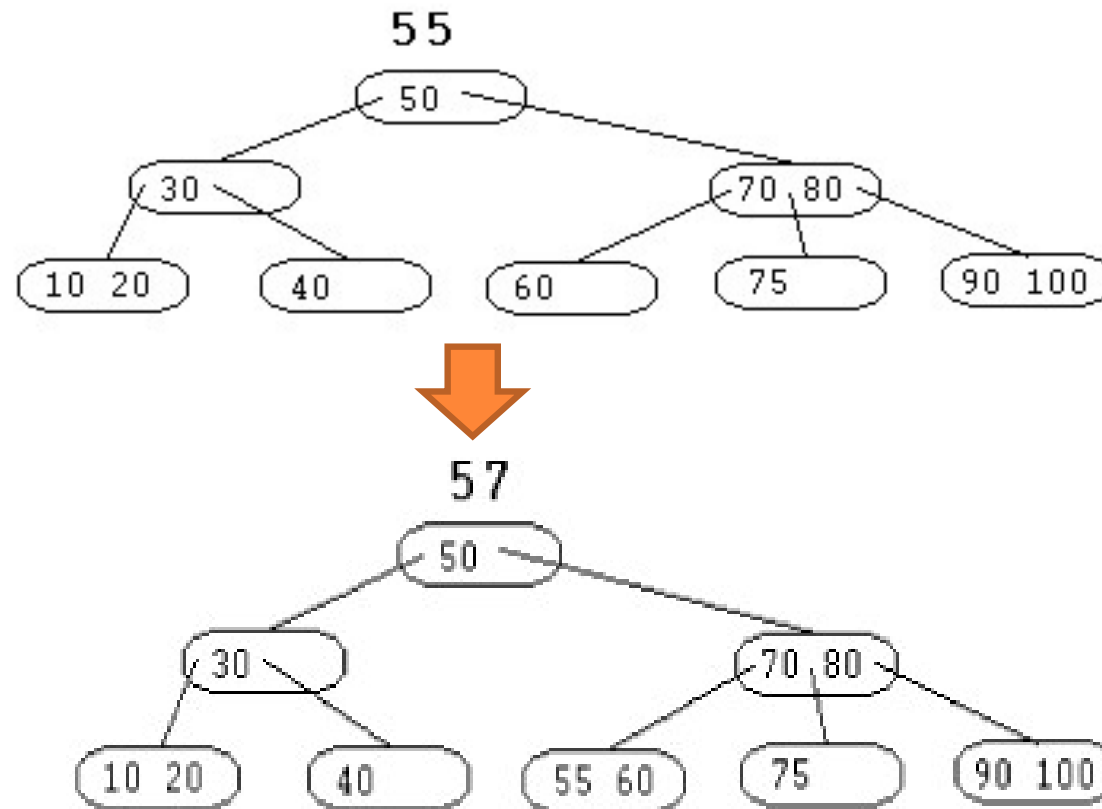
- ❖ Ainsi, dans un B-arbre, le taux de remplissage de chaque nœud est maintenu entre 50% et 100%
- ❖ Les insertions et les suppressions deviennent simples à réaliser sauf dans deux cas particuliers:
 - Insertion dans un nœud déjà plein
 - Suppression dans un nœud entraînant un taux de remplissage inférieur à 50%

V. INSERTION

- ❖ **L'insertion se fait toujours au niveau des feuilles.**
- ❖ Ainsi, l'insertion se déroule comme suit:
 1. Rechercher la position de l'élément à insérer, soit P la feuille trouvée.
 2. Si la feuille P n'est pas pleine alors insérer la clé à sa bonne position.

V. INSERTION

❖ **Exemple:** Considérons le B-arbre suivant d'ordre 3, insérer les valeurs suivantes dans l'ordre : 55, 57, 95, 85, 87.

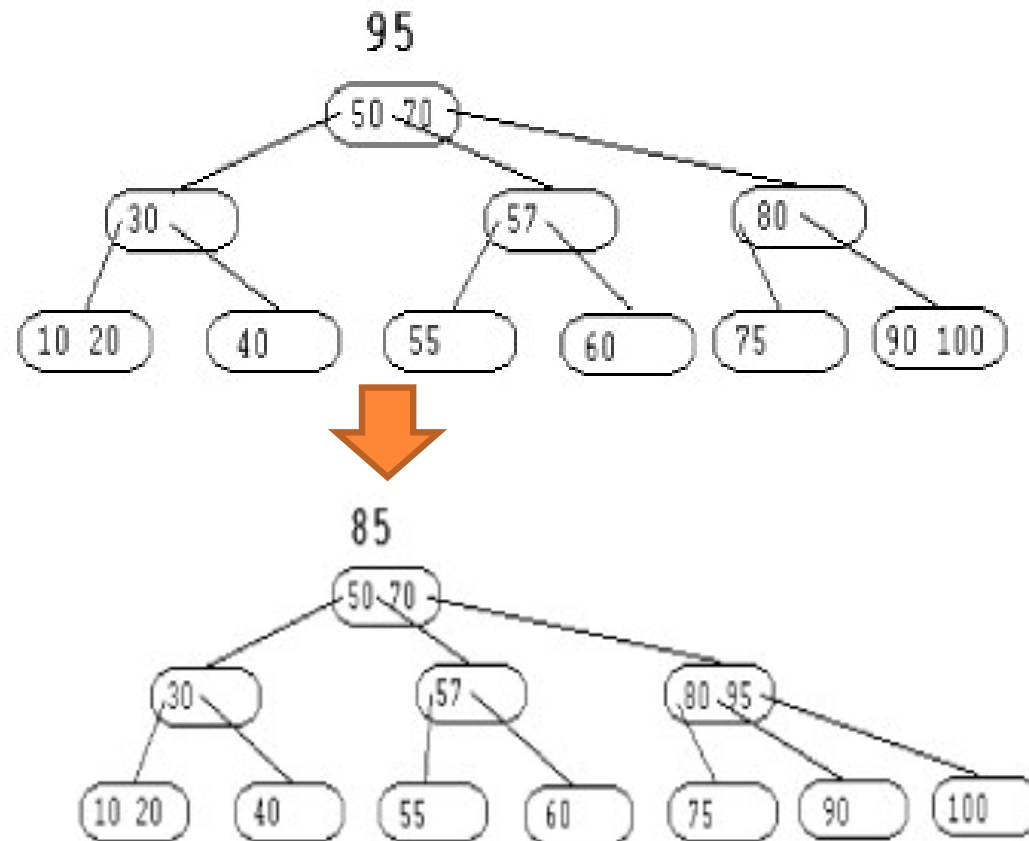


V. INSERTION

3. Sinon, la feuille P est pleine, l'éclatement se fait en cascade (approche ascendante) comme suit
 - a. Classer les clés dans l'ordre croissant : $k_1, k_2, \dots, k_m$; soit k_{mil} la clé du milieu
 - b. Laisser les d plus petites clés ($k_1 \dots k_{mil-1}$) dans le nœud P
 - c. Déplacer les d plus grandes clés ($k_{mil+1} \dots k_m$) dans un nouveau nœud (Q)
 - d. Remonter la clé médiane (ou la valeur du milieu k_{mil}) dans le nœud père de telle sorte que le nœud P se trouvera à sa gauche et Q à sa droite.
 - e. Soit P le nœud père, aller à 2 (application récursive de ce principe éventuellement jusqu'à la racine)

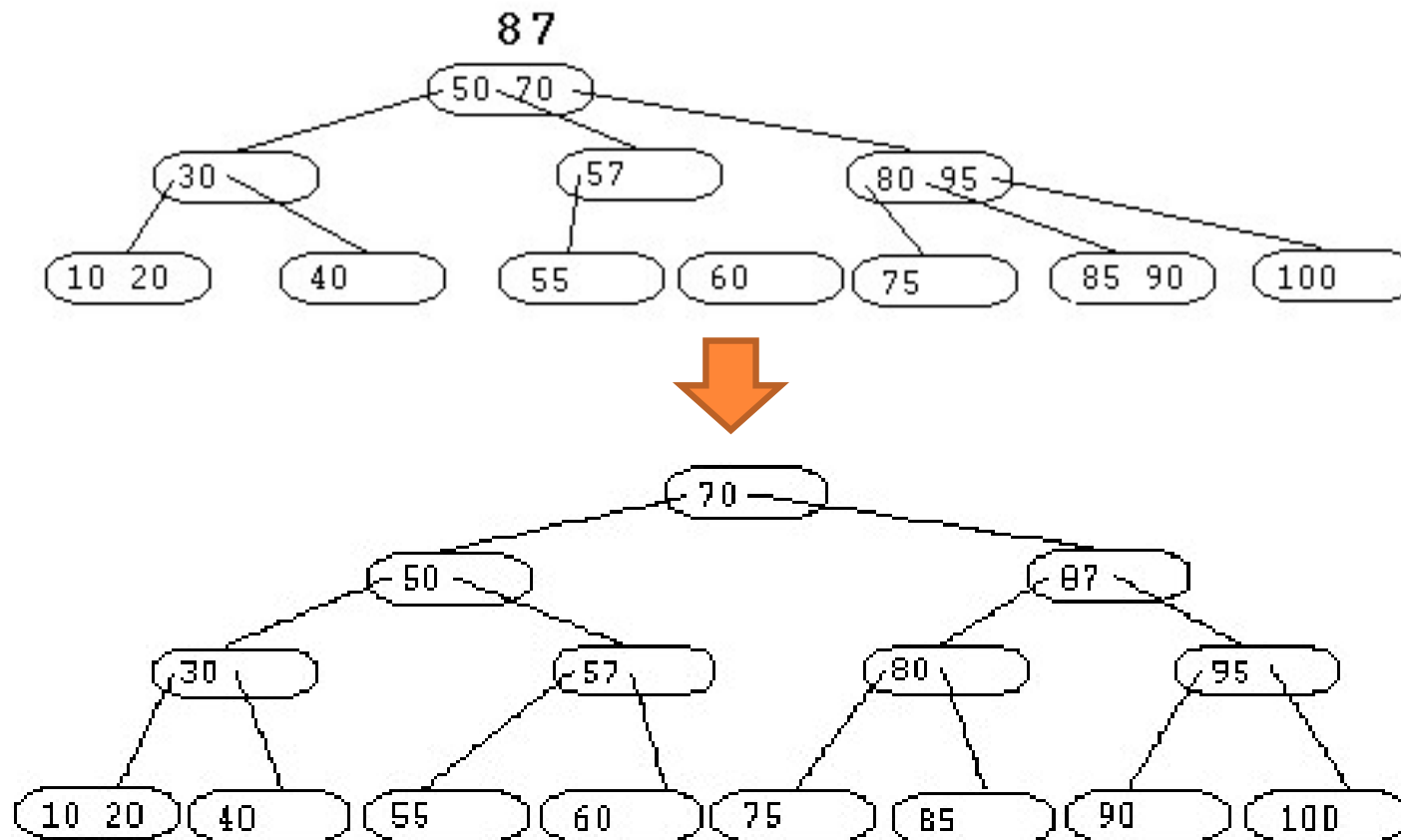
V. INSERTION

❖ **Exemple:** Considérons le B-arbre suivant d'ordre 3, insérer les valeurs suivantes dans l'ordre : 55, 57, 95, 85, 87.



V. INSERTION

❖ **Exemple:** Considérons le B-arbre suivant d'ordre 3, insérer les valeurs suivantes dans l'ordre : 55, 57, 95, 85, 87.



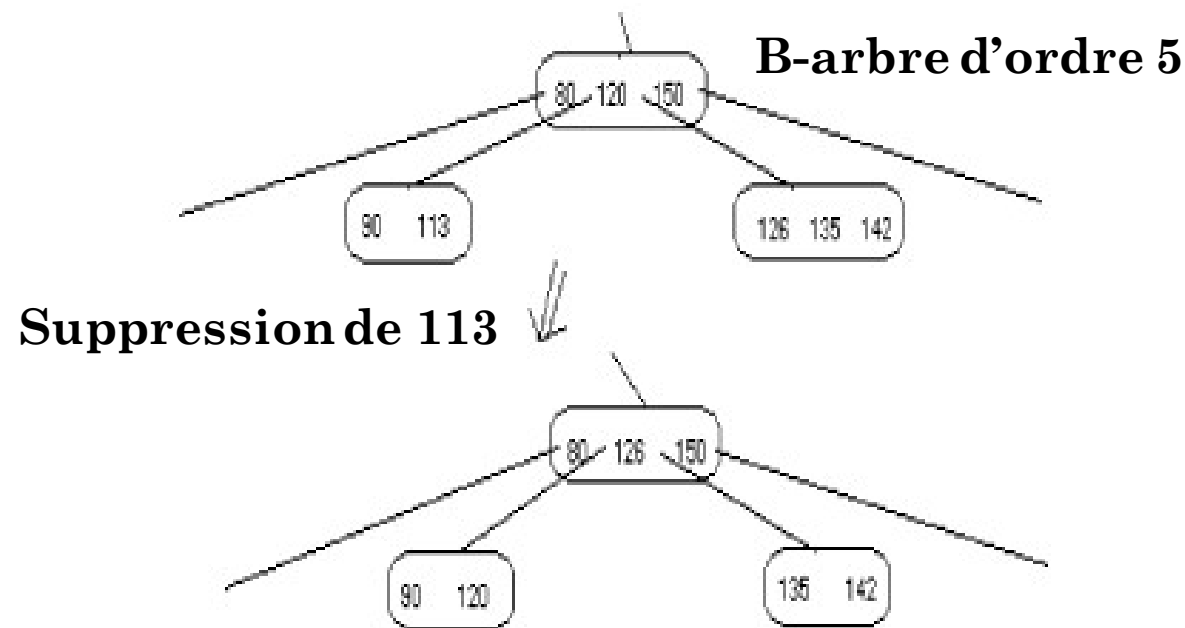
V. SUPPRESSION

❖ Il faut supprimer l'élément tout en préservant la qualité de B-arbre, c'est à dire en gardant au moins d clés dans le nœud (non racine).

❖ C'est le cas de la suppression dans un Arbre de M-aire Recherche (AMR). En plus, si le nœud feuille qui contenait la clé à supprimer a moins de d clés, alors l'action suivante est entreprise :

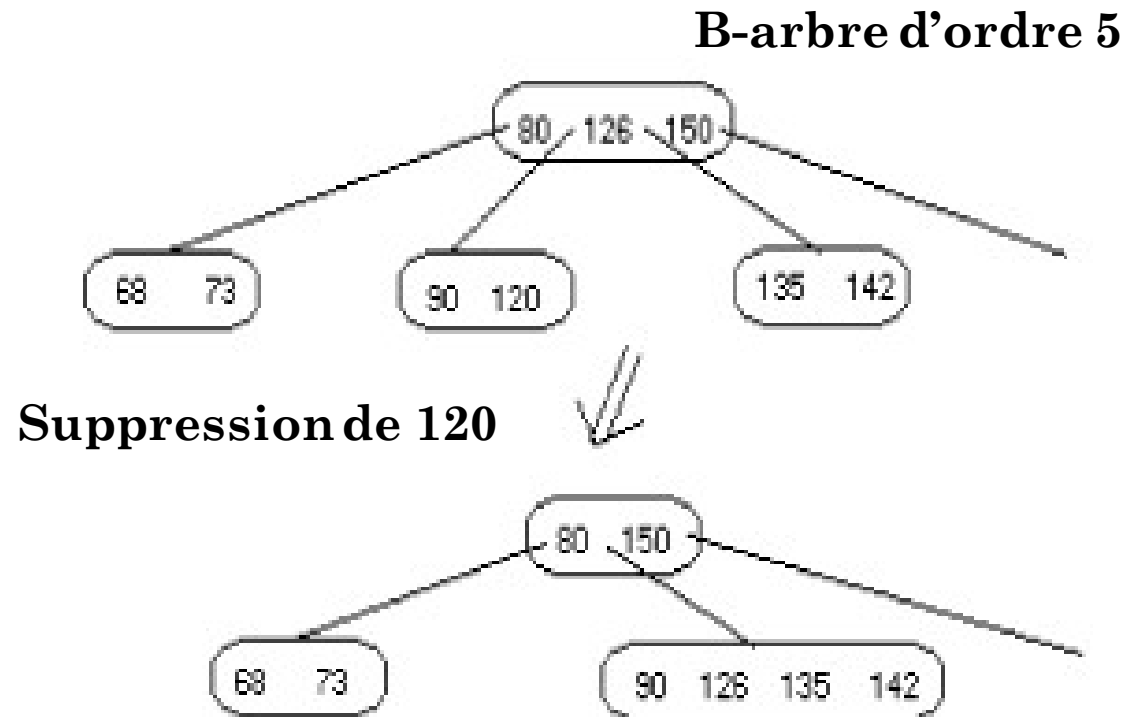
V. SUPPRESSION

➤ **Cas 1:** Si l'un des frères (gauche ou droit) contient plus de d clés, alors la clé, soit K_s , dans le nœud père qui sépare entre les deux frères est ajoutée au nœud "underflow" et le dernier (si frère gauche) ou le premier élément (si frère droit) est ajoutée au père à la place de K_s .



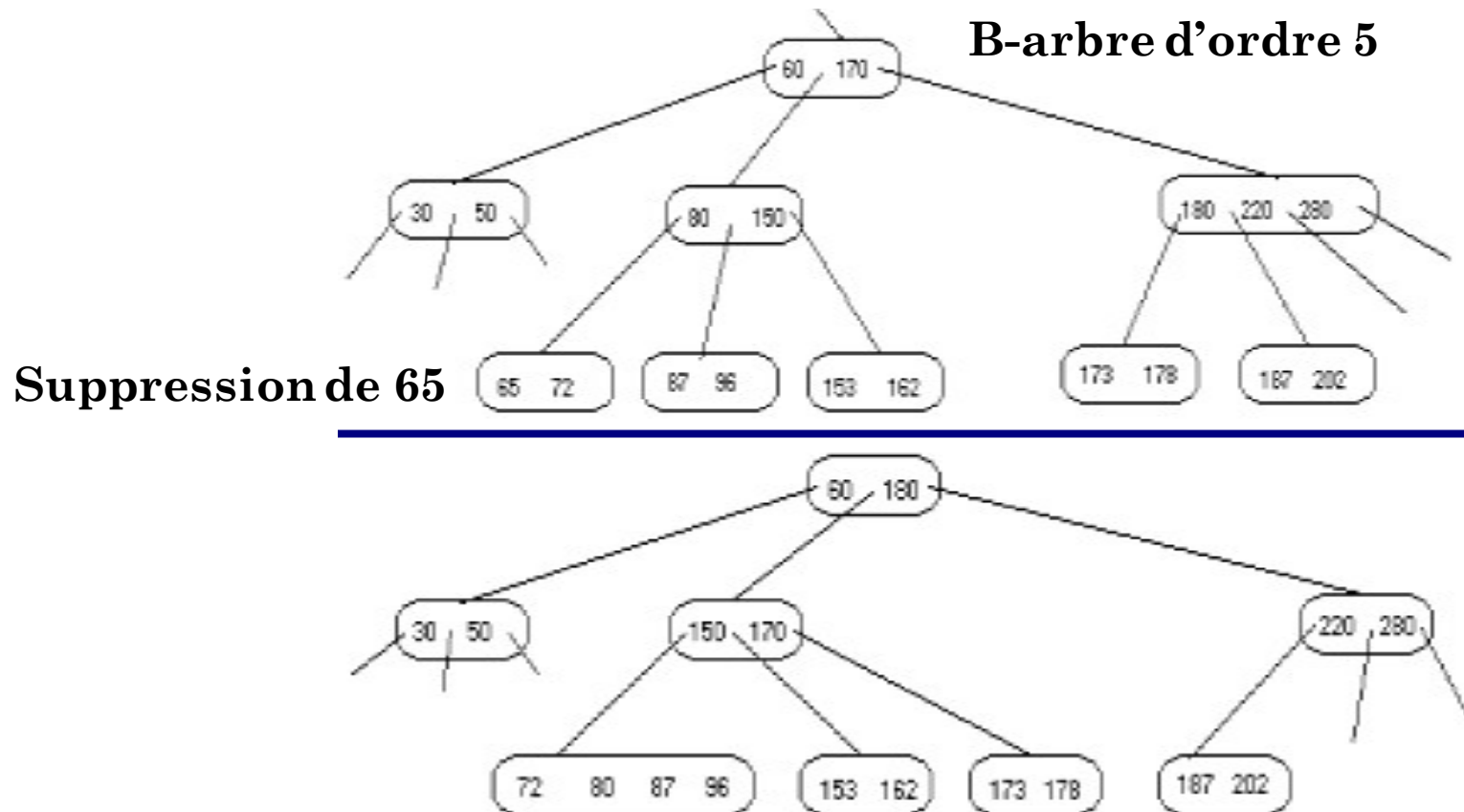
V. SUPPRESSION

➤ **Cas 2-a:** Si les deux frères contiennent exactement d clés, le nœud "underflow" et l'un de ses frères seront concaténés (fusionnés ou consolidés) en un seul nœud qui contient aussi la clé séparatrice de leur père.



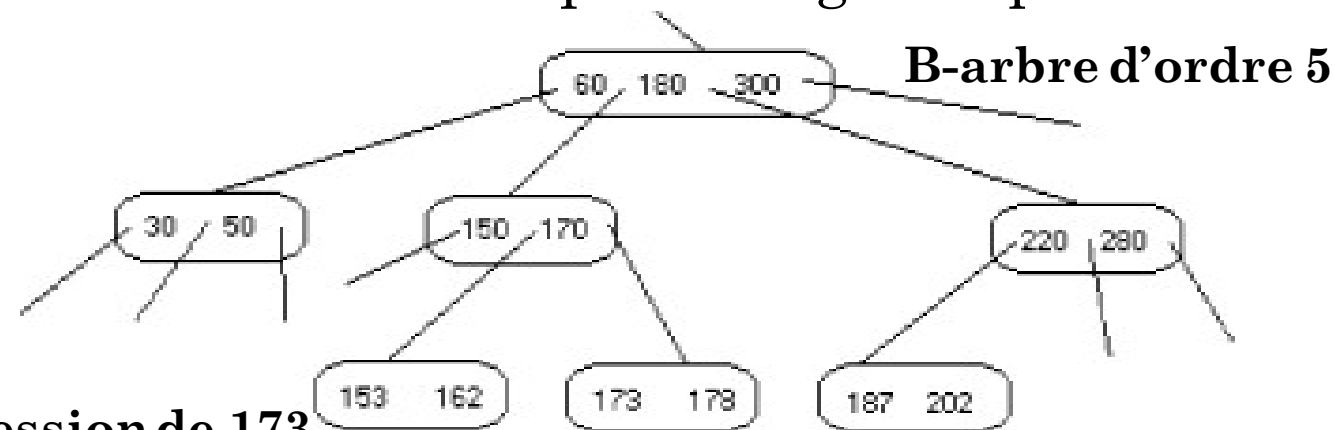
V. SUPPRESSION

➤ **Cas 2-b:** Il est aussi possible que le père contienne seulement des clés et par conséquent il n'a pas de clé à donner. Dans ce cas, il peut emprunter de son père et frère.

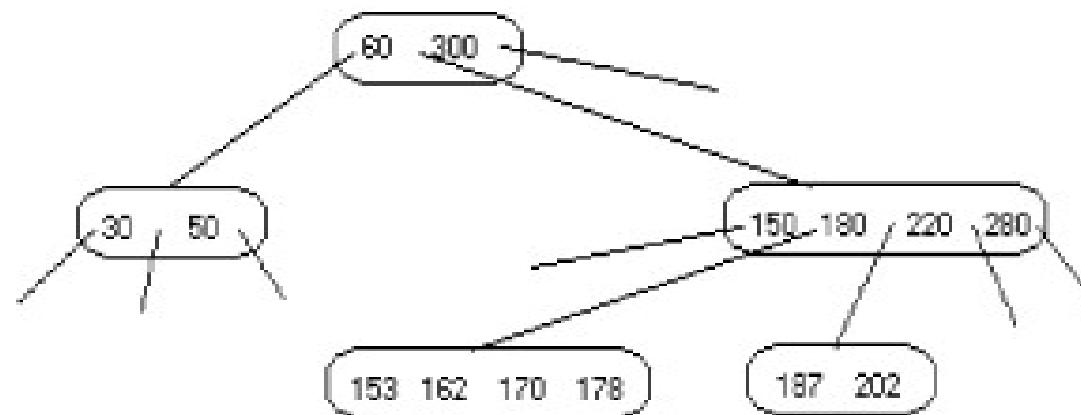


V. SUPPRESSION

➤ **Cas 2-c:** Dans le pire des cas, quand les frères du père n'ont pas des clés à donner, le père et son frère peuvent aussi être concaténés et une clé est prise du grand père.



Suppression de 173



V. SUPPRESSION

➤ **Cas 2-d:** Si tous les antécédents d'un nœud et leurs frères contiennent exactement d clés, une clé sera prise de la racine (due aux concaténations en cascades):

- Si la racine avait plus d'une clé, ceci termine le processus.
- Si la racine contenait une seule clé, elle sera utilisée dans la concaténation. Le nœud racine est libéré et la profondeur de l'arbre est réduit d'une unité.

V. B-ARBRE

❖ Exercice:

1. Construire un B-arbre d'ordre 3 et un B-arbre d'ordre 5 à partir des valeurs suivantes:

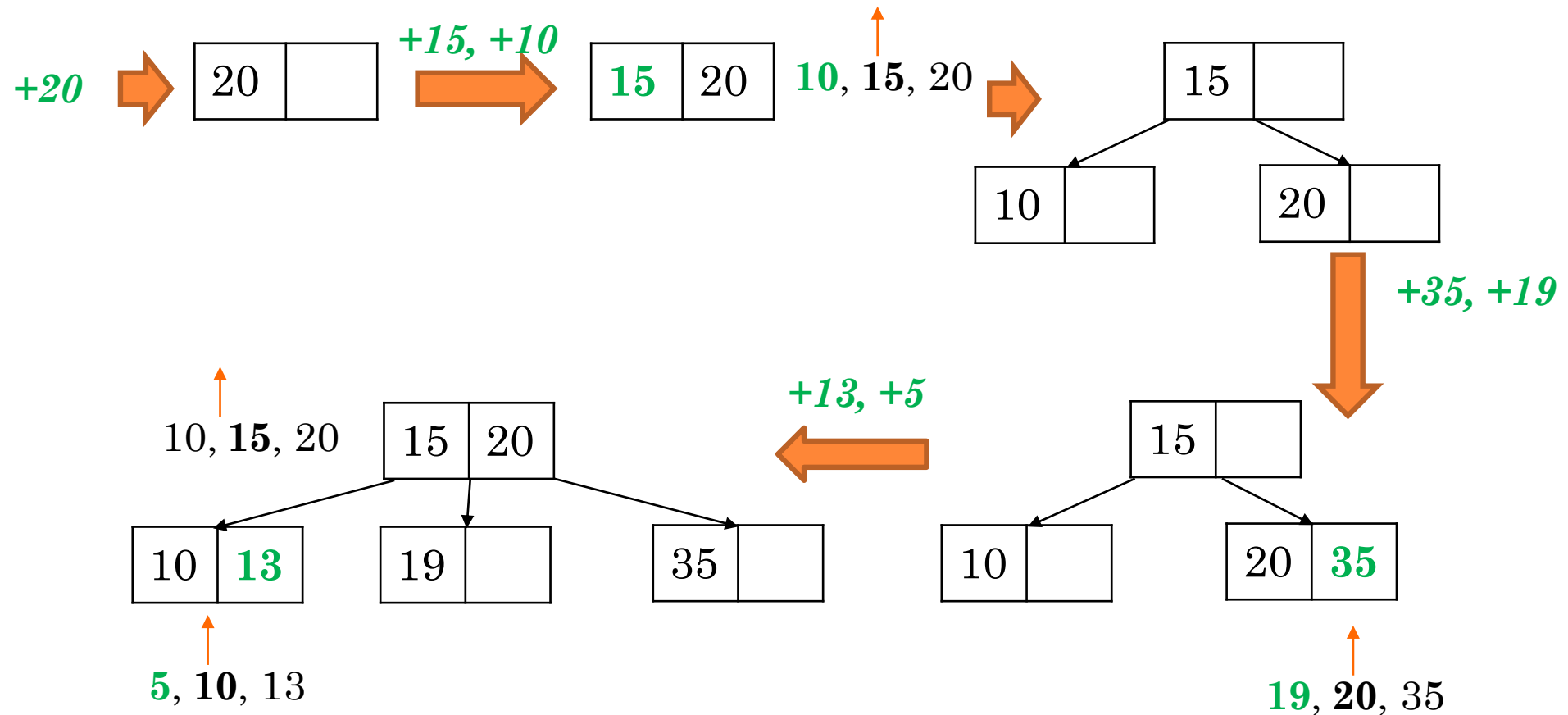
20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

2. Supprimer du B-arbre d'ordre 5 les valeurs suivantes: 5, 15, 25, 35, 13, 10.
3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10, 7, 5, 13, 25, 38.

V. B-ARBRE

1. Construction de B-arbre d'ordre 3

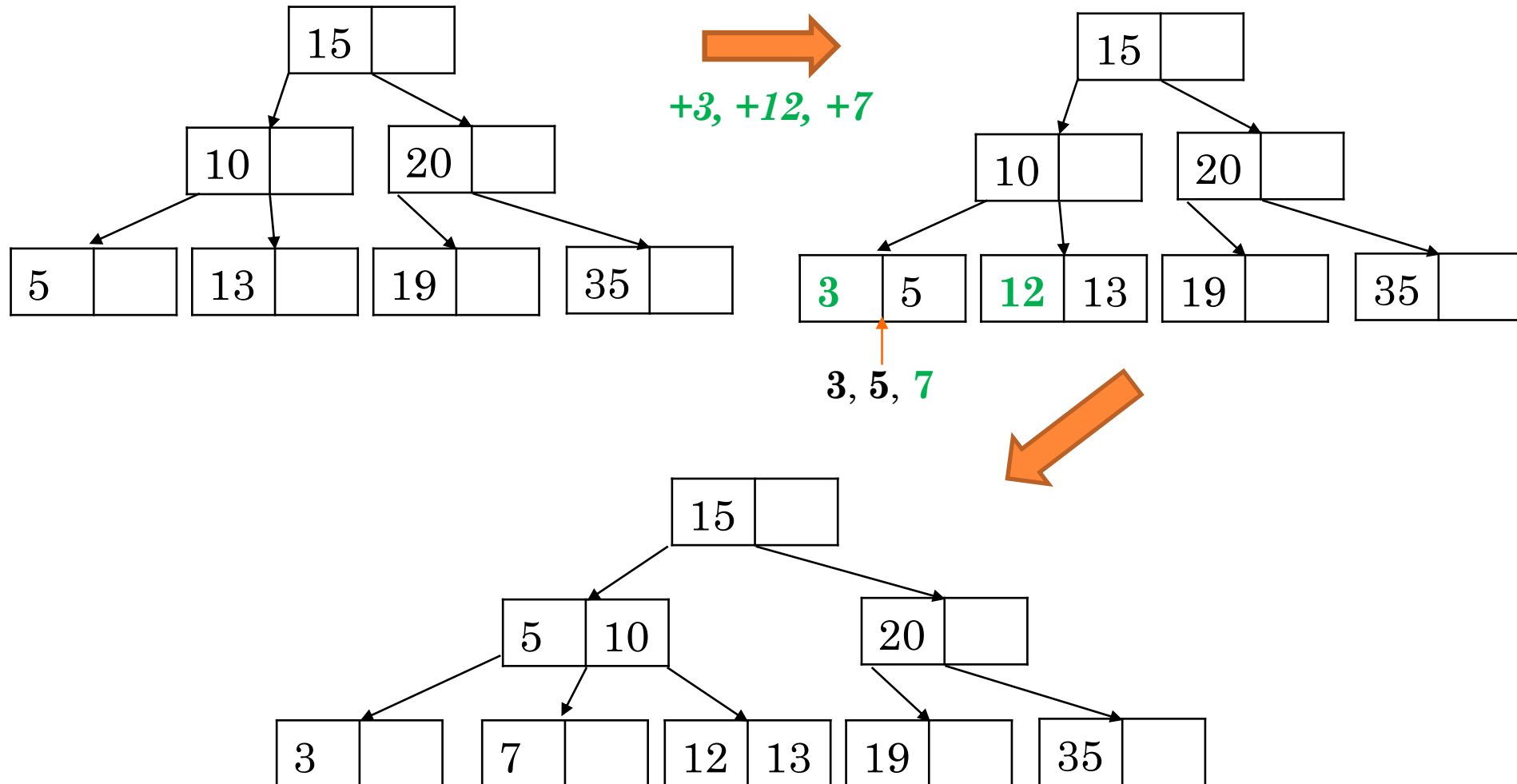
20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----



V. B-ARBRE

1. Construction de B-arbre d'ordre 3

20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

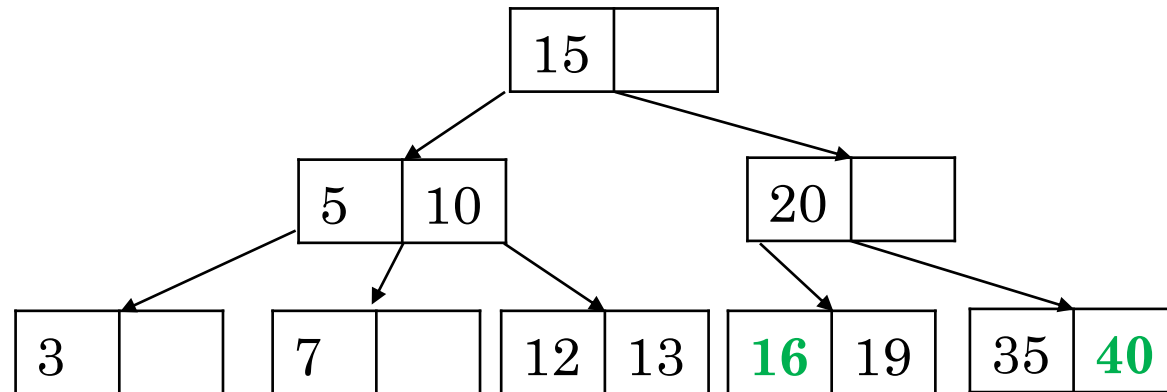


V. B-ARBRE

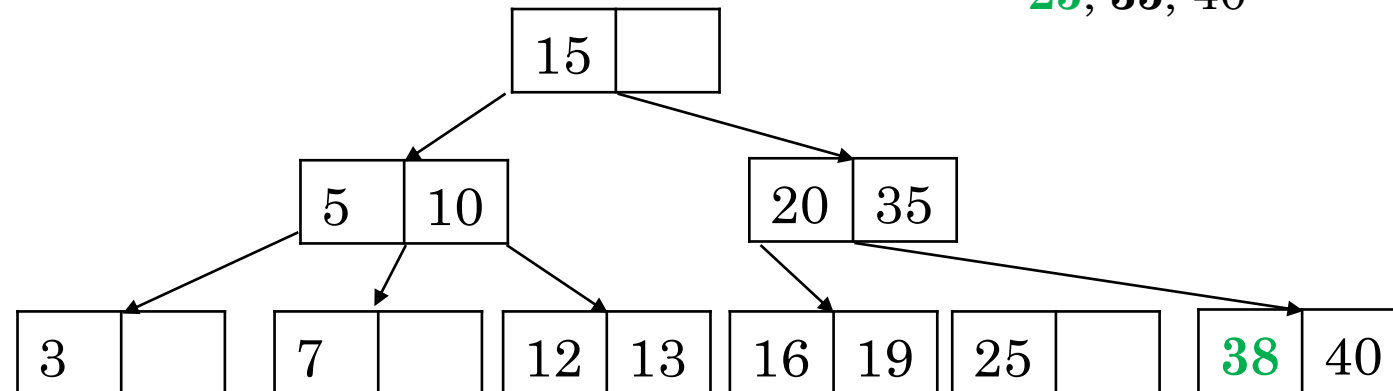
1. Construction de B-arbre d'ordre 3

20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

➔
+16, +40, +25



+38
↓
25, 35, 40



V. B-ARBRE

1. Construction de B-arbre d'ordre 5

20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

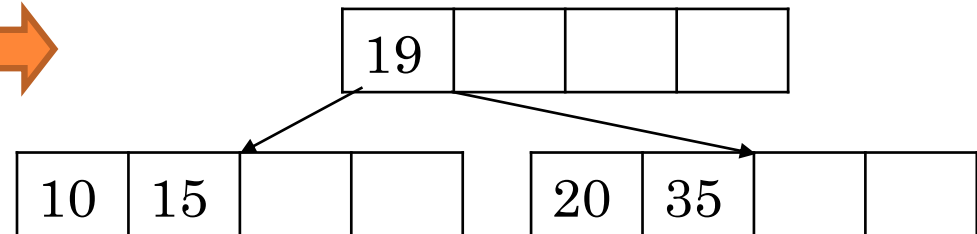
+20, +15,
+10, +35,
+19



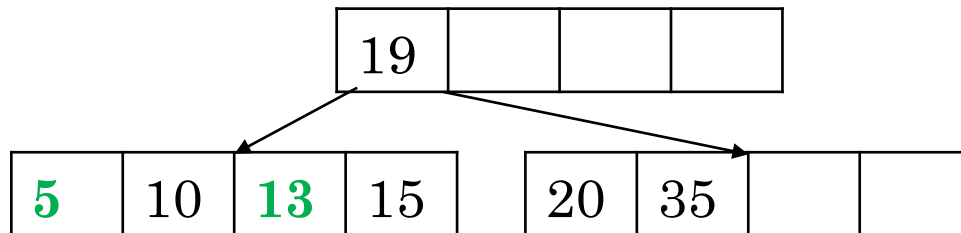
10	15	20	35
----	----	----	----



10, 15, 19, 20, 35



+13, +5, +3

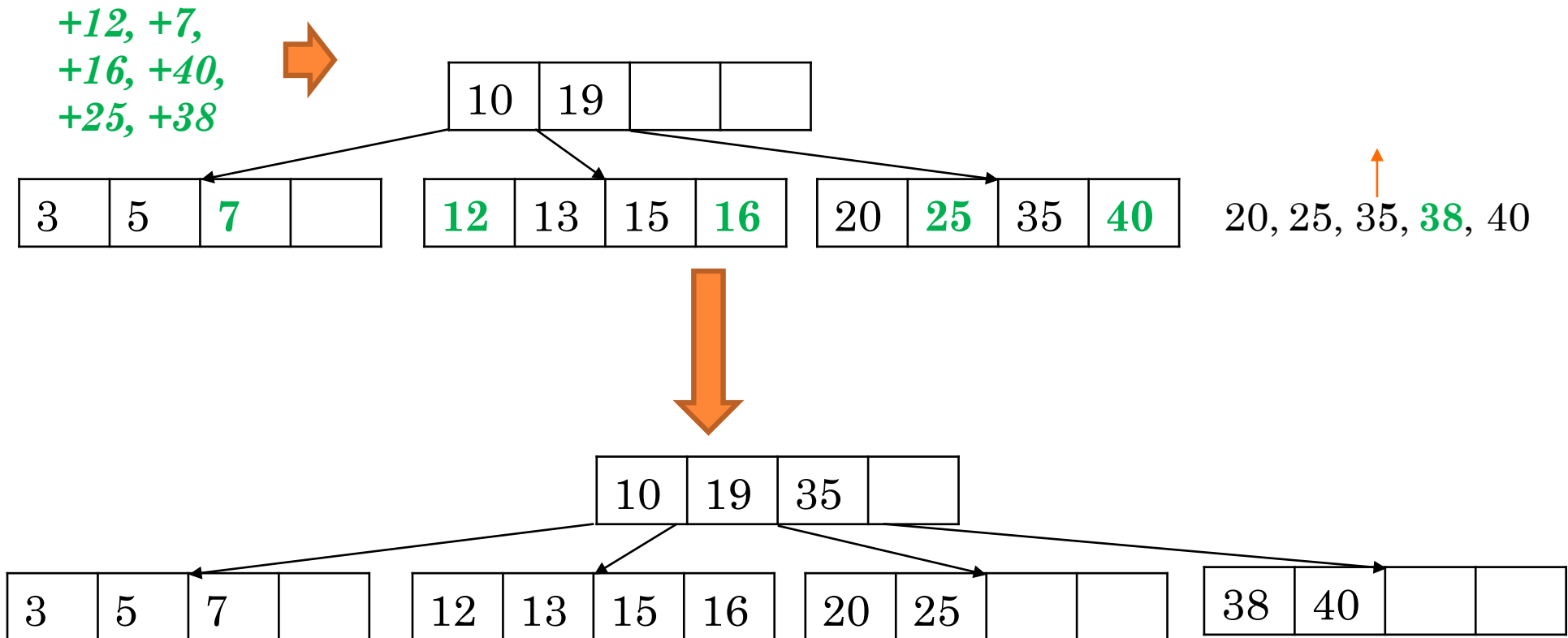


3, 5, 10, 13, 15



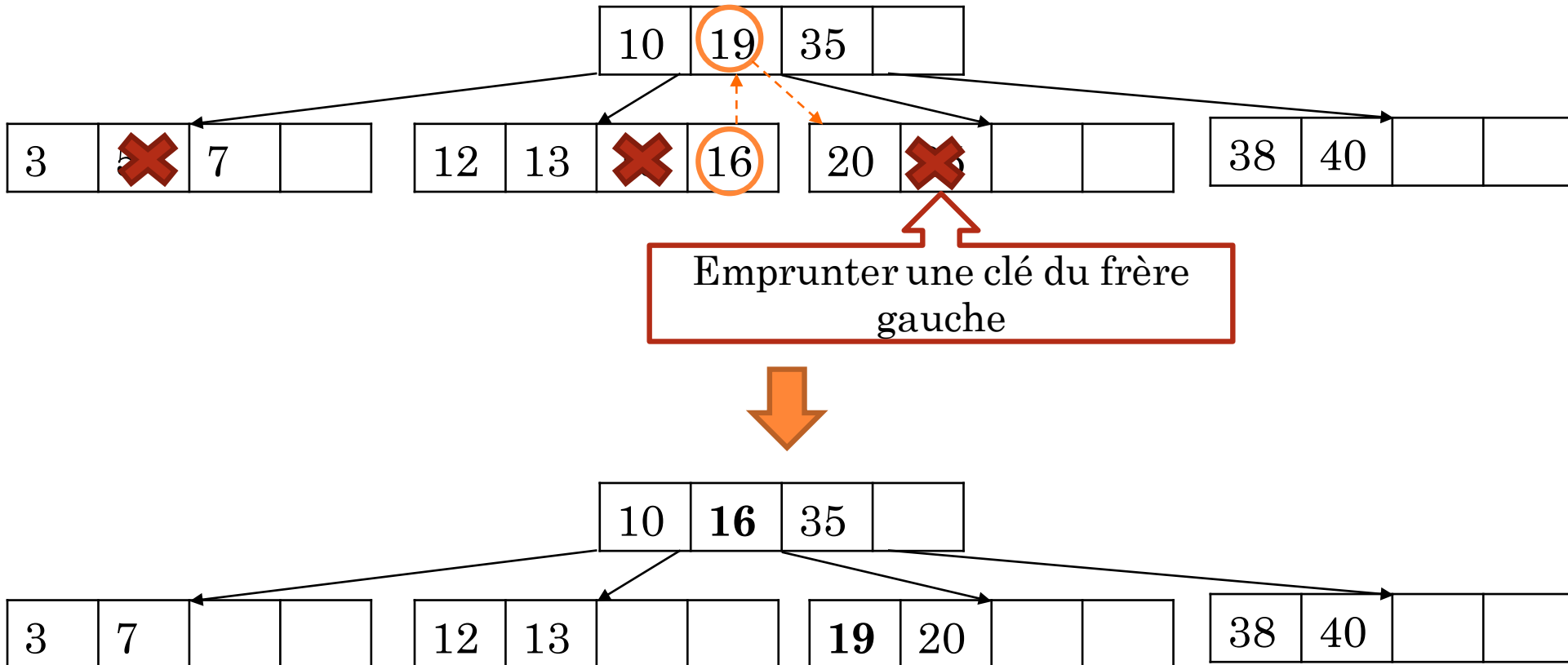
V. B-ARBRE

1. Construction de B-arbre d'ordre 5



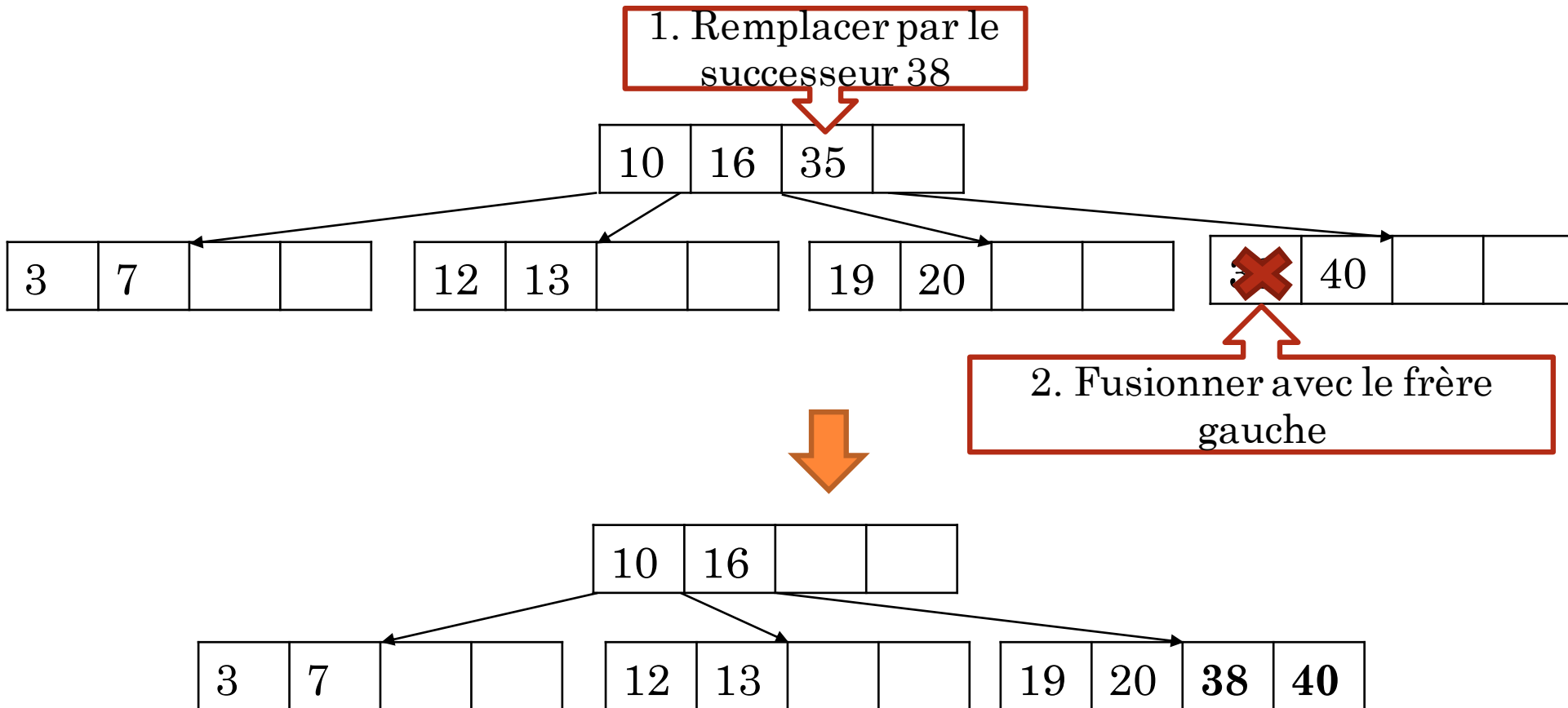
V. B-ARBRE

2. Supprimer du B-arbre d'ordre 5 les valeurs suivantes: 5, 15, 25, 35, 13, 10.



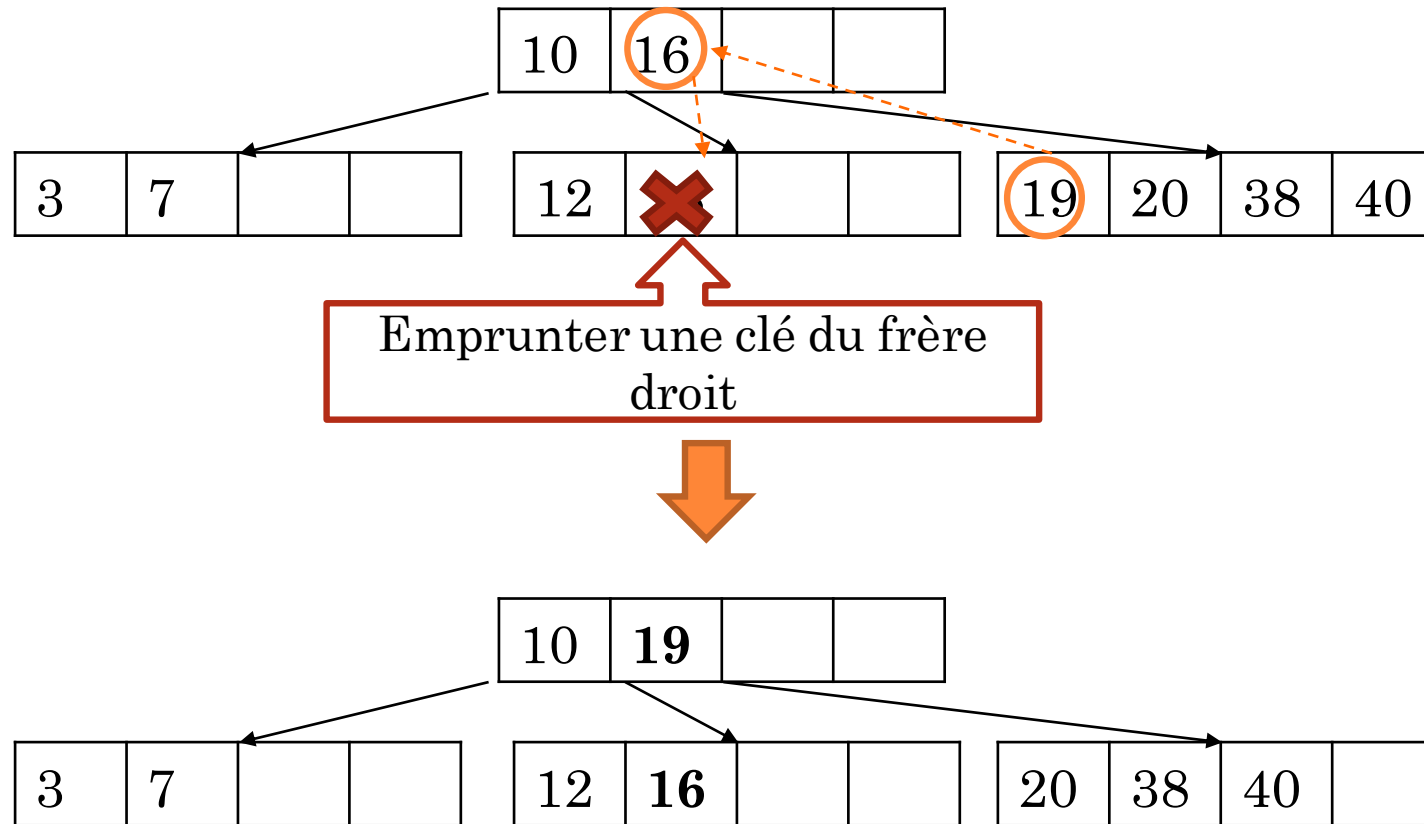
V. B-ARBRE

2. Supprimer du B-arbre d'ordre 5 les valeurs suivantes: 5, 15, 25, 35, 13, 10.



V. B-ARBRE

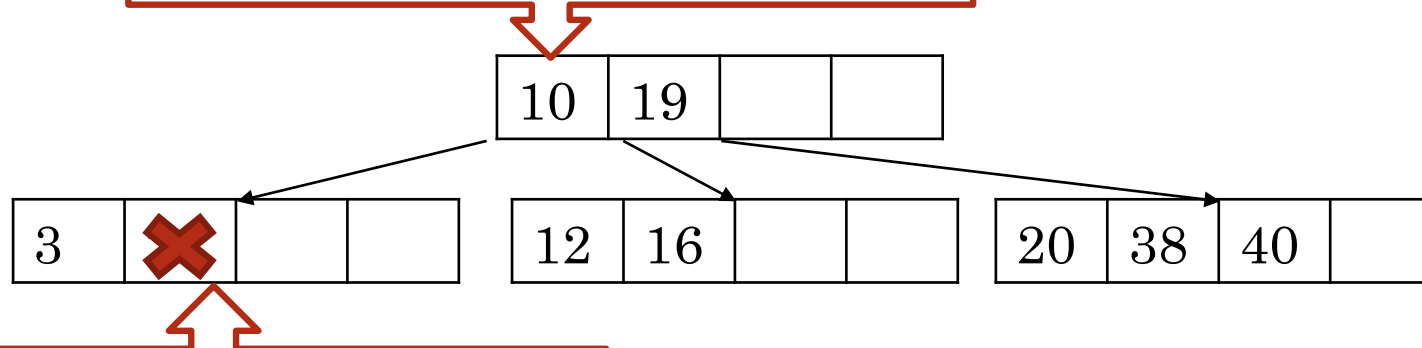
2. Supprimer du B-arbre d'ordre 5 les valeurs suivantes: 5, 15, 25, 35, 13, 10.



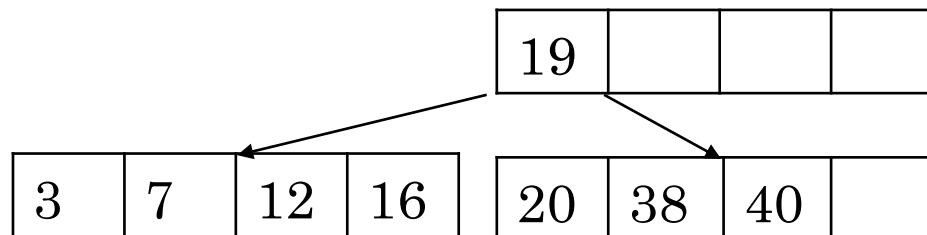
V. B-ARBRE

2. Supprimer du B-arbre d'ordre 5 les valeurs suivantes: 5, 15, 25, 35, 13, 10.

1. Remplacer par le prédécesseur 7

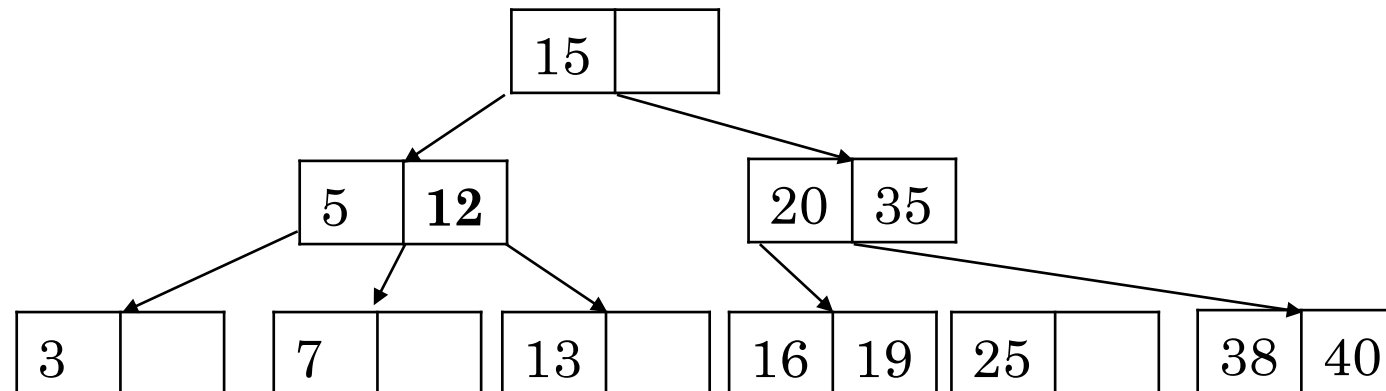
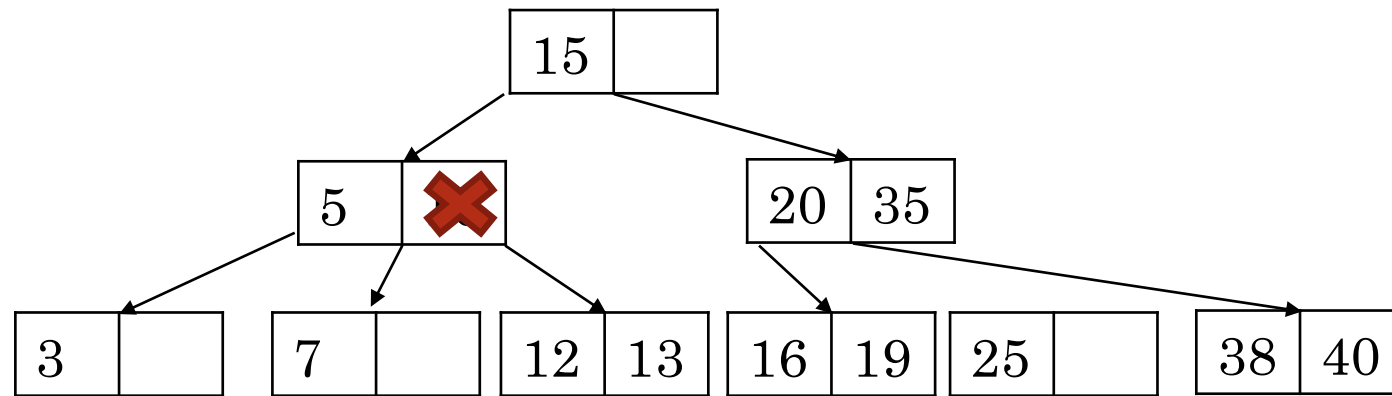


2. Fusionner avec le frère droit



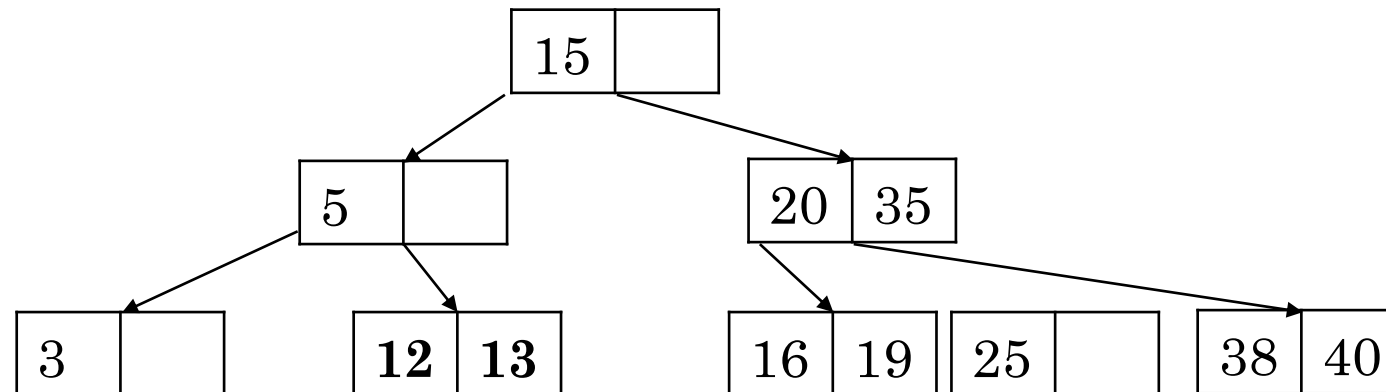
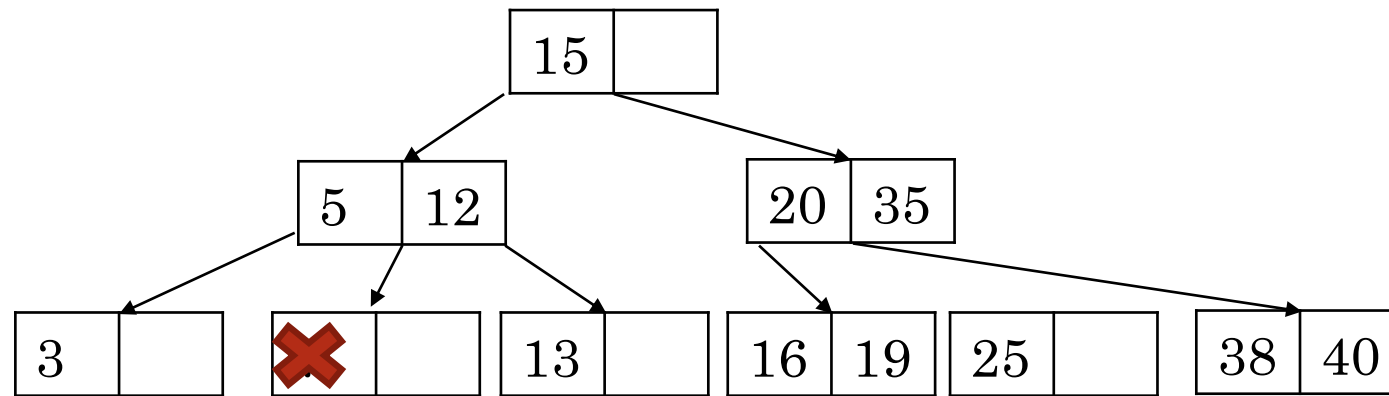
V. B-ARBRE

3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10 (remplacer par le successeur 12), 7, 5, 13, 25, 38.



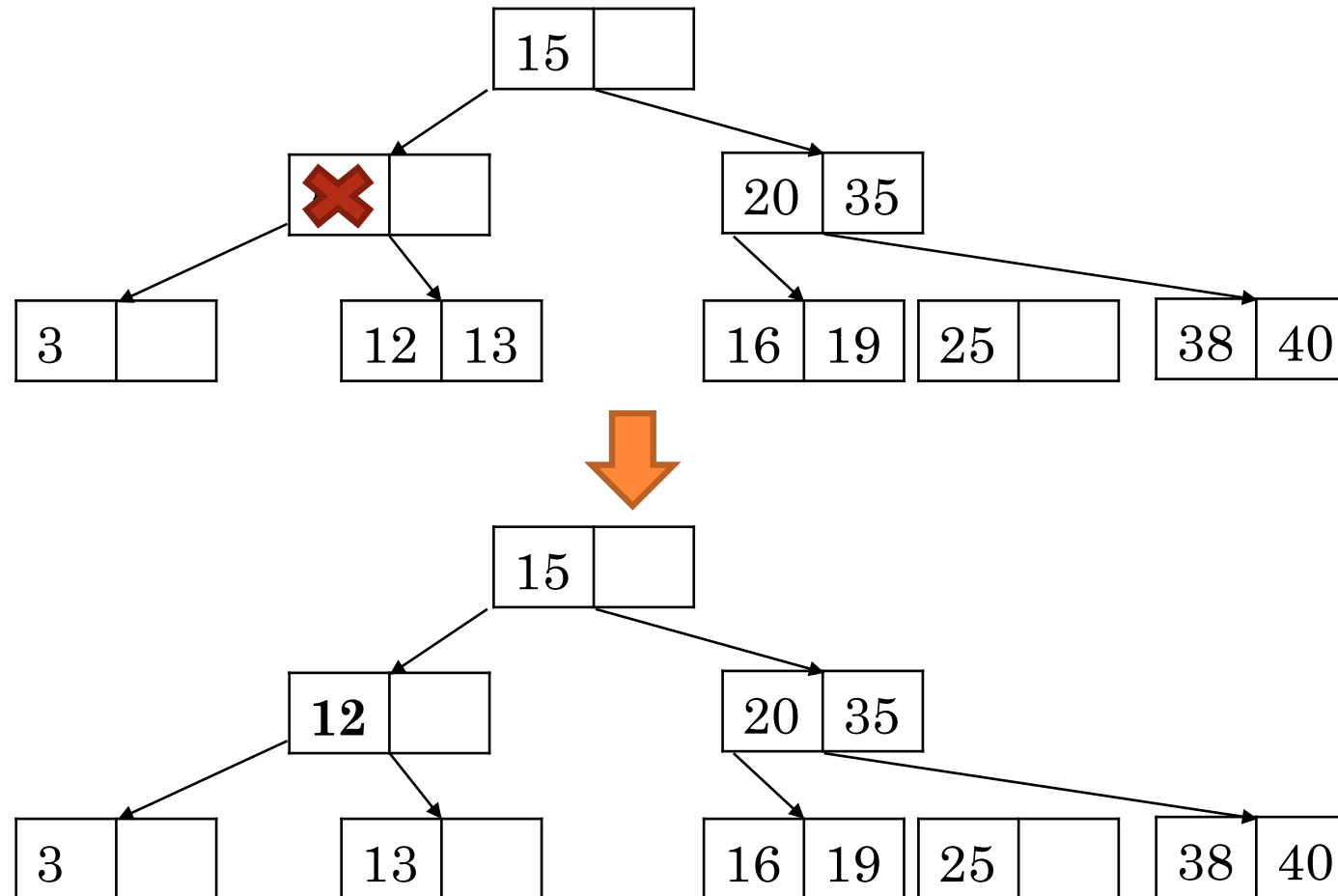
V. B-ARBRE

3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10, 7(fusionner avec le frère droit), 5, 13, 25, 38.



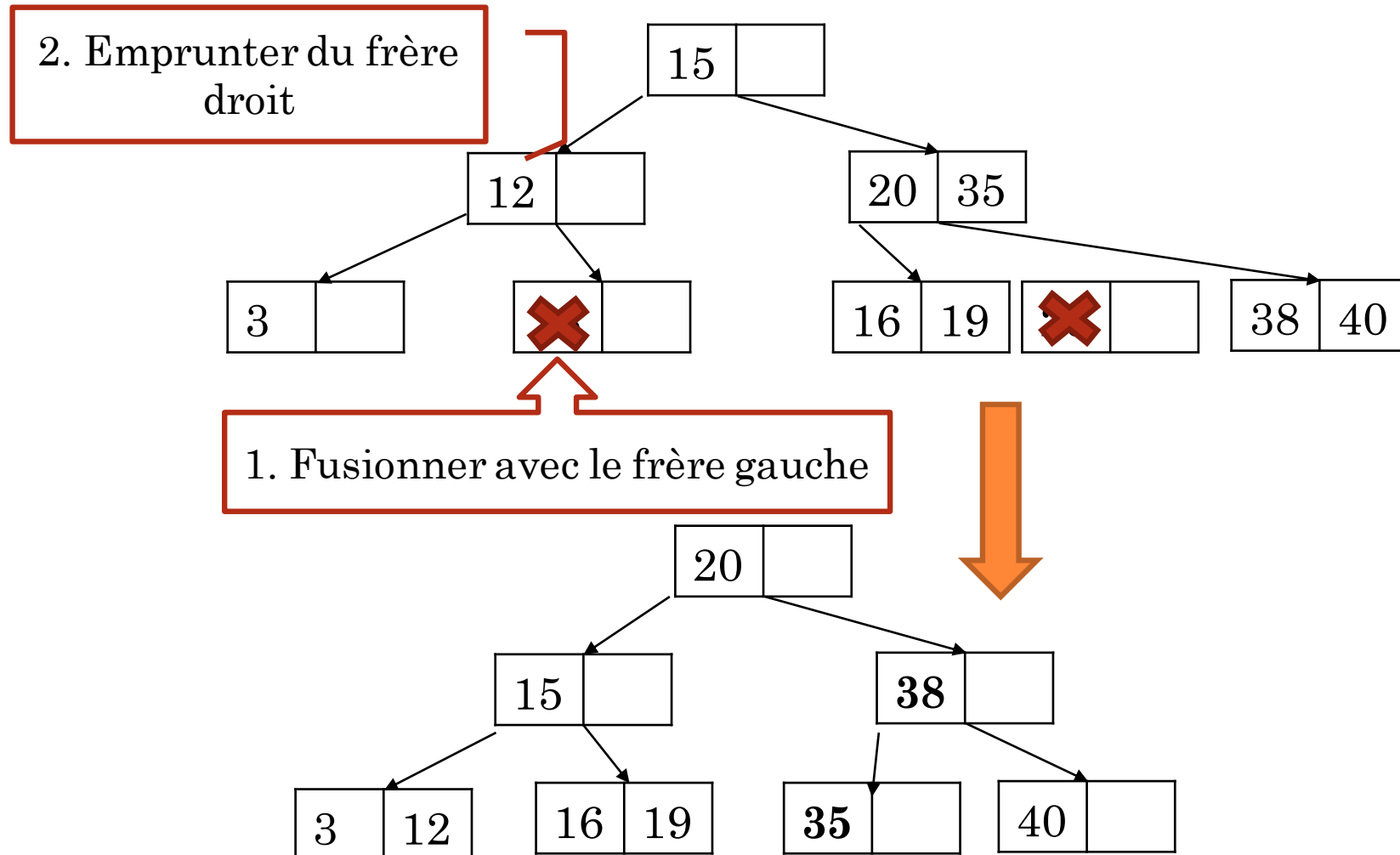
V. B-ARBRE

3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10, 7, 5(remplacer par le successeur 12), 13, 25, 38.



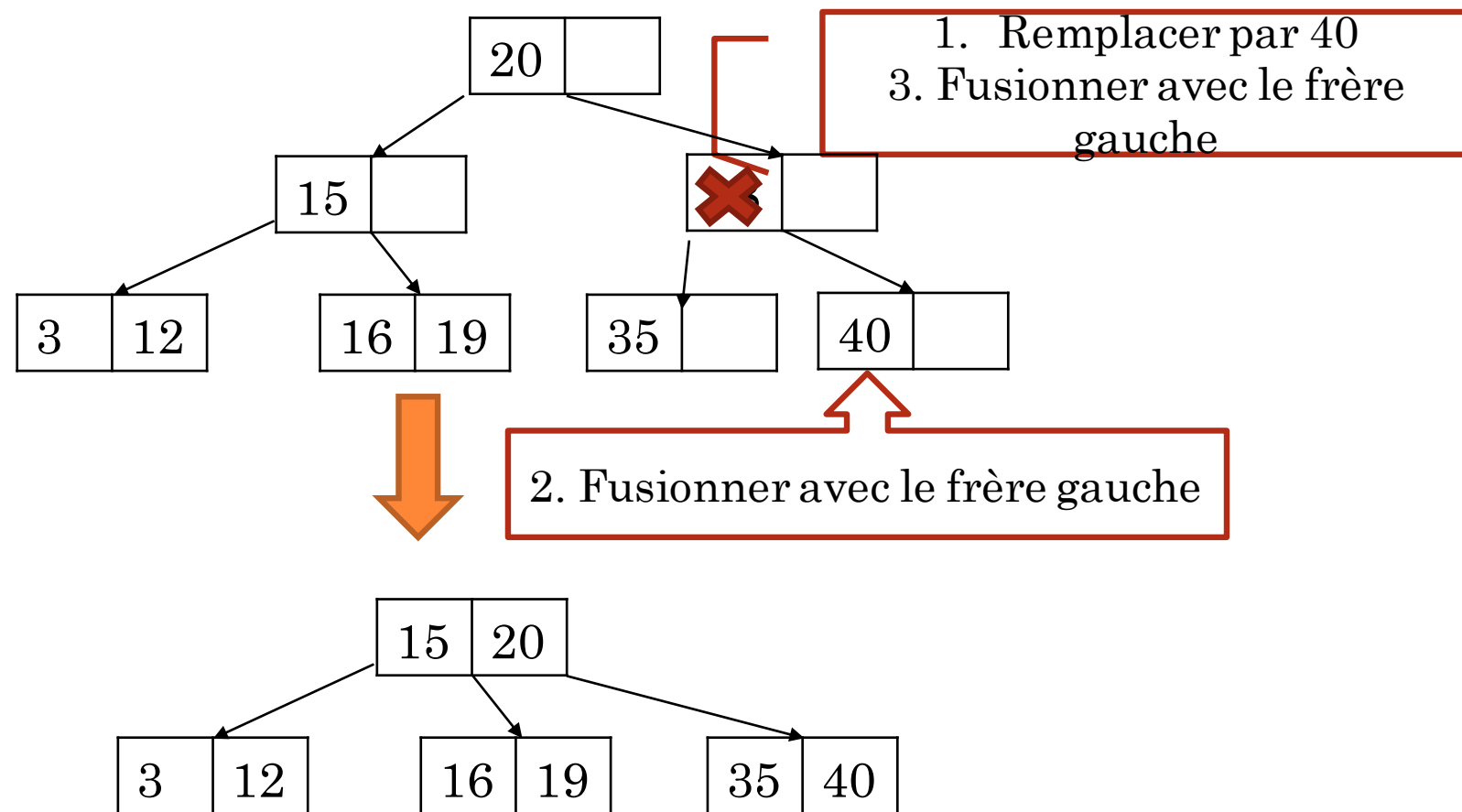
V. B-ARBRE

3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10, 7, 5, 13, 25 (emprunter du frère droit), 38.



V. B-ARBRE

3. Supprimer du B-arbre d'ordre 3 les valeurs suivantes: 10, 7, 5, 13, 25, 38.



SOURCES DE CE COURS

- N. EL-ALLIA , Cours d'Algorithmique et Structures de données dynamiques, Ecole nationale Supérieure d'Informatique (ESI), 2014.
- Djamel Eddine ZEGOUR, Cours de Structures de Données, Ecole nationale Supérieure d'Informatique (ESI), Disponible sur http://zegour.esi.dz/Cours/Cours_sdd.htm
- W. K. Hidouci, Cours Structures De Données et Fichiers, École nationale Supérieure d'Informatique, Disponible sur <http://hidouci.esi.dz/algo/>
- A. Walid, Gestion des fichiers, Disponible sur <https://slideplayer.fr/slide/9332040/>
- A. Aroussi, Cours d' Algorithmique Avancée, Université de Blida 1, 2016, Disponible sur <https://sites.google.com/a/esi.dz/informatiqueblida/Algorithmique/annee-universitaire-2015-2016>

III.TAS: HAUTEUR

❖ **Théorème:** Un TAS de n nœud a une hauteur $O(\log_2 n)$

❖ **Démonstration**

➤ Soit h , la hauteur d'un tas de n nœud

➤ Au niveau $i \neq h$, $n_i = 2^i$

➤ Donc $n = 2^0 + 2^1 + 2^2 + \dots + 2^{h-1} + c$ tel que , $0 \leq c \leq 2^h$

$$n = 2^h + c \geq 2^h \Rightarrow h \leq \log_2 n \Rightarrow O(h) = O(\log_2 n).$$

❖ **Conséquence:** Les opérations proportionnelles à h sont $O(\log_2 n)$

III.TAS: INSERTION

- ❖ L'insertion d'une valeur « v » peut nécessiter $O(h) = O(\log_2(n))$ opérations où h est la hauteur du TAS et n est son nombre des nœuds.
- ❖ En effet, au pire des cas, l'échange peut se remonter à la racine dans le cas où « v » est inférieure à la valeur de la racine, ainsi, il devient le nouveau minimum

RECHERCHE

- ❖ La recherche d'une valeur « x » dans un TAS peut nécessiter $O(n)$ opérations où n est le nombre des nœuds.
 - ❖ *Pire des cas: Si « x » est supérieure à la valeur maximale du TAS_{min} [ou inférieure à la valeur minimale du TAS_{max}], alors on doit descendre jusqu'au dernier niveau en parcourant tous les nœuds pour vérifier que la valeur recherchée n'existe pas.*
- ❖ La recherche étant un pré-requis à la suppression d'une valeur « x », une suppression peut nécessiter $O(n)$ opérations aussi.

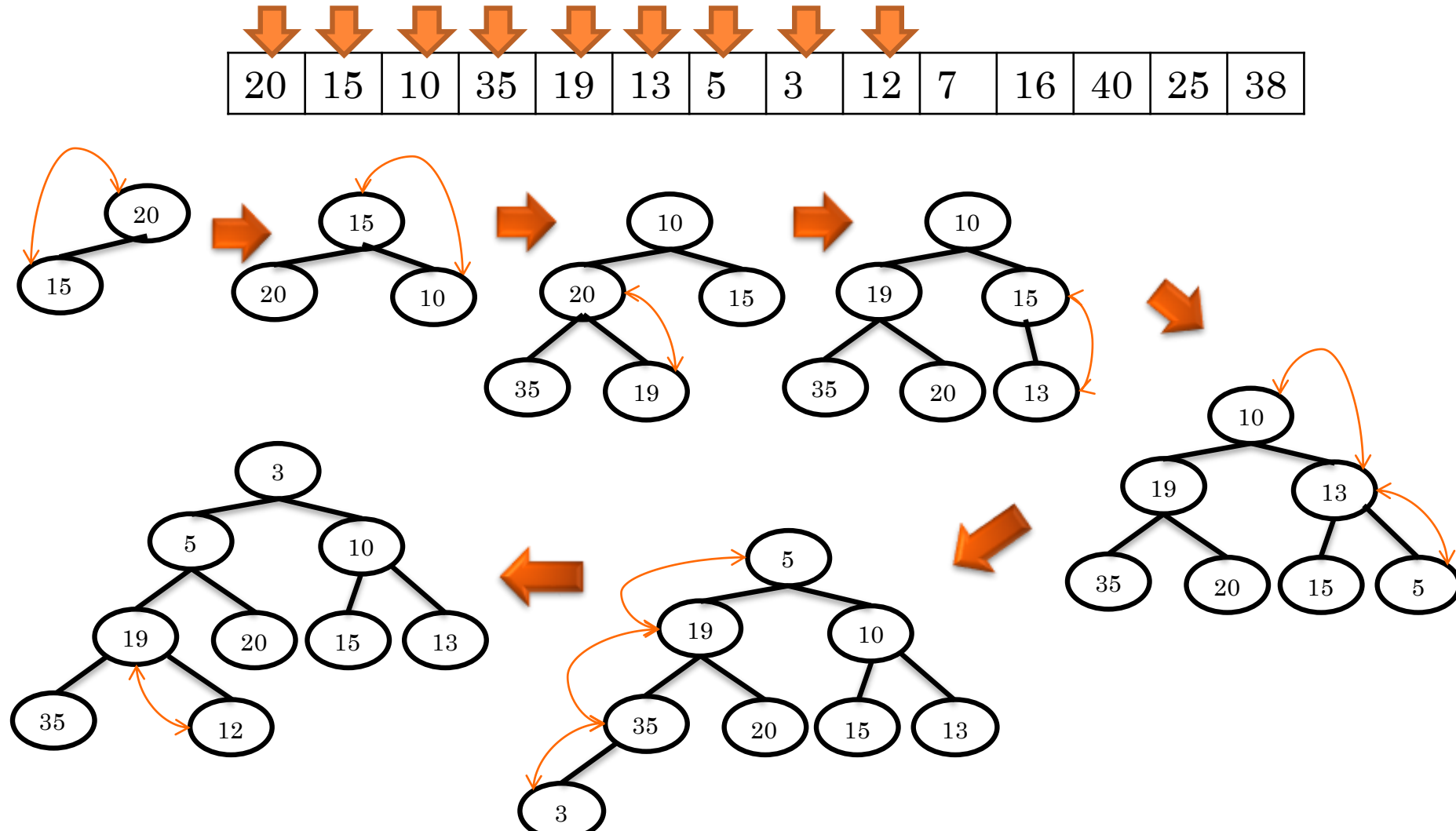
III.TAS: INSERTION

❖ **Exercice:** Construire un $TAS_{\min}$ et un $TAS_{\max}$ à partir des valeurs suivantes:

20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

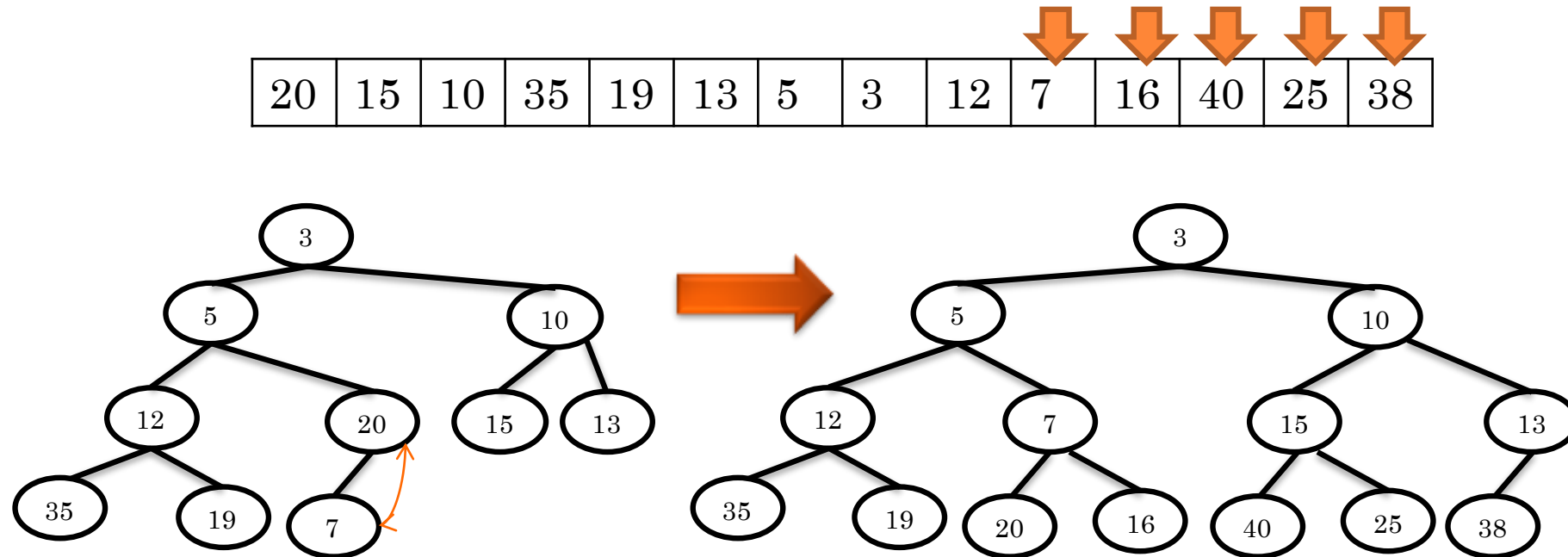
III.TAS: INSERTION

- ❖ **Solution:** Construire un $TAS_{\min}$ à partir des valeurs suivantes:



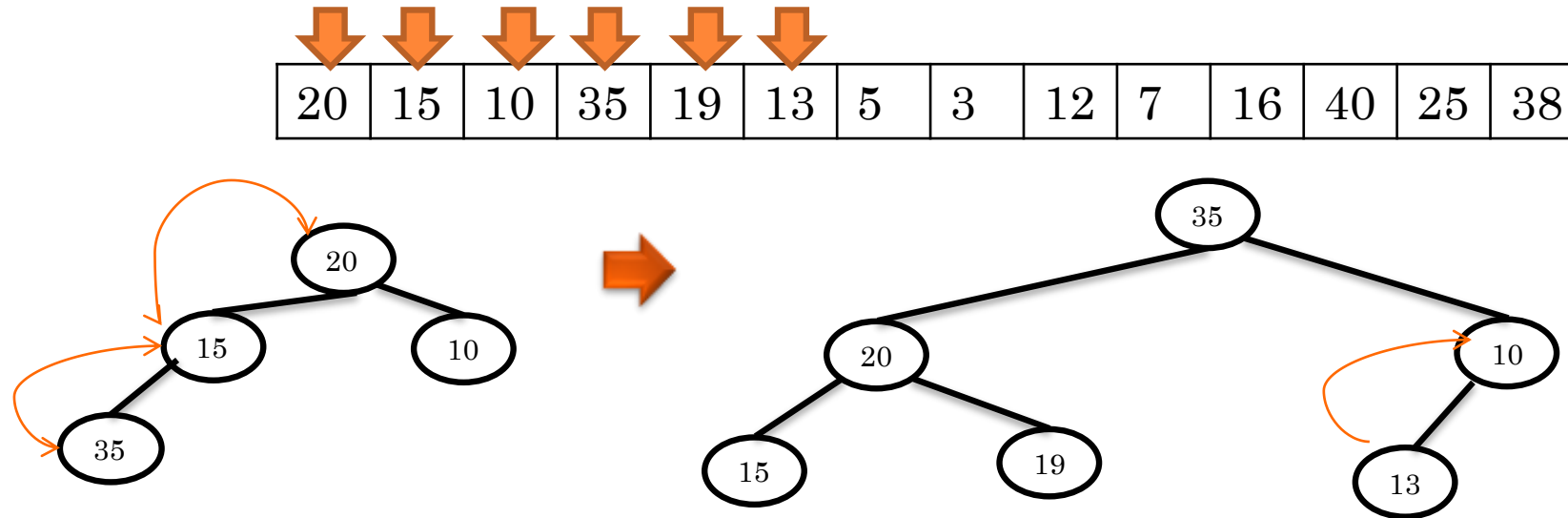
III.TAS: INSERTION

- ❖ **Solution:** Construire un $TAS_{\min}$ à partir des valeurs suivantes:



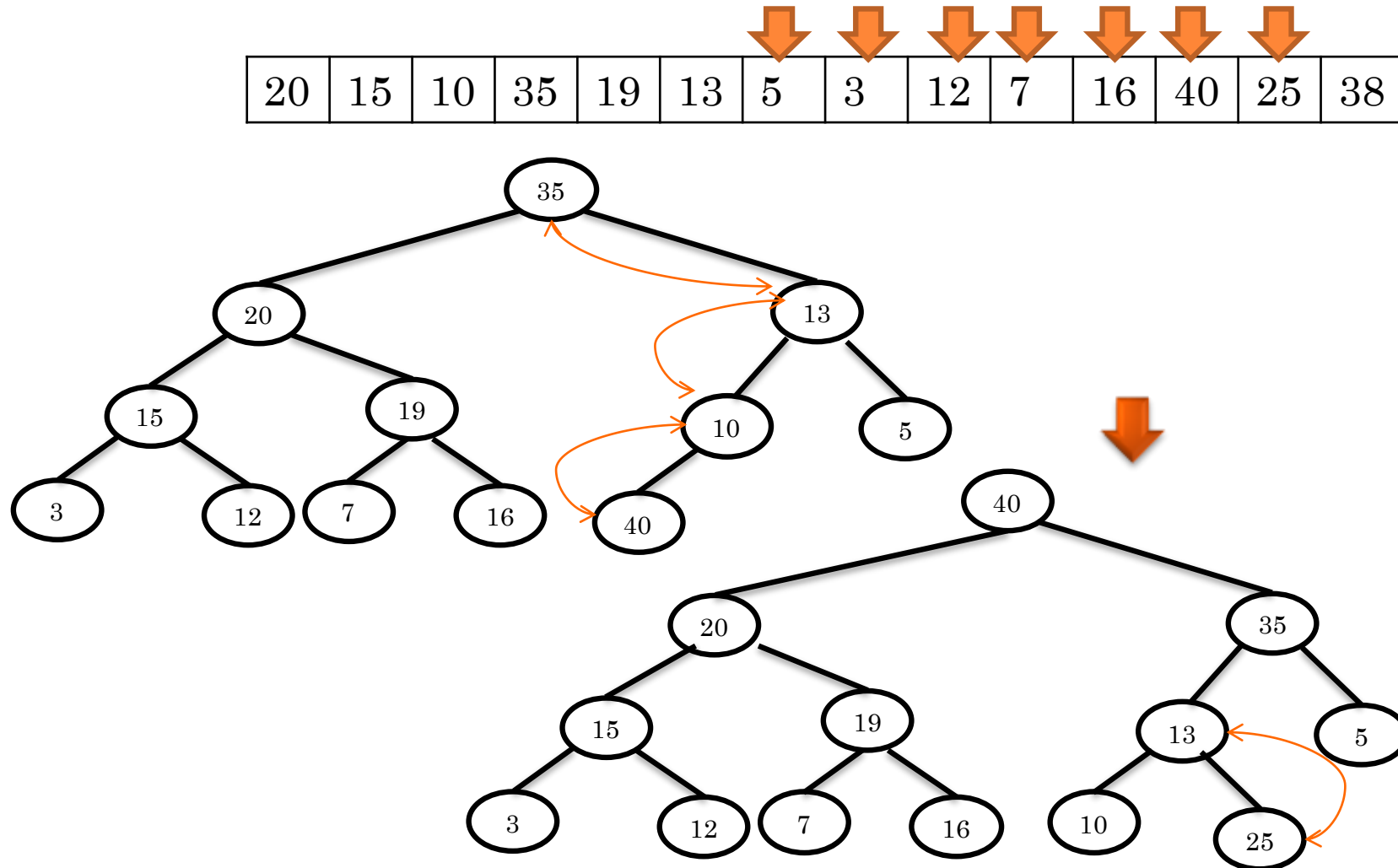
III.TAS: INSERTION

- ❖ **Solution:** Construire un $TAS_{\max}$ à partir des valeurs suivantes:



III.TAS: INSERTION

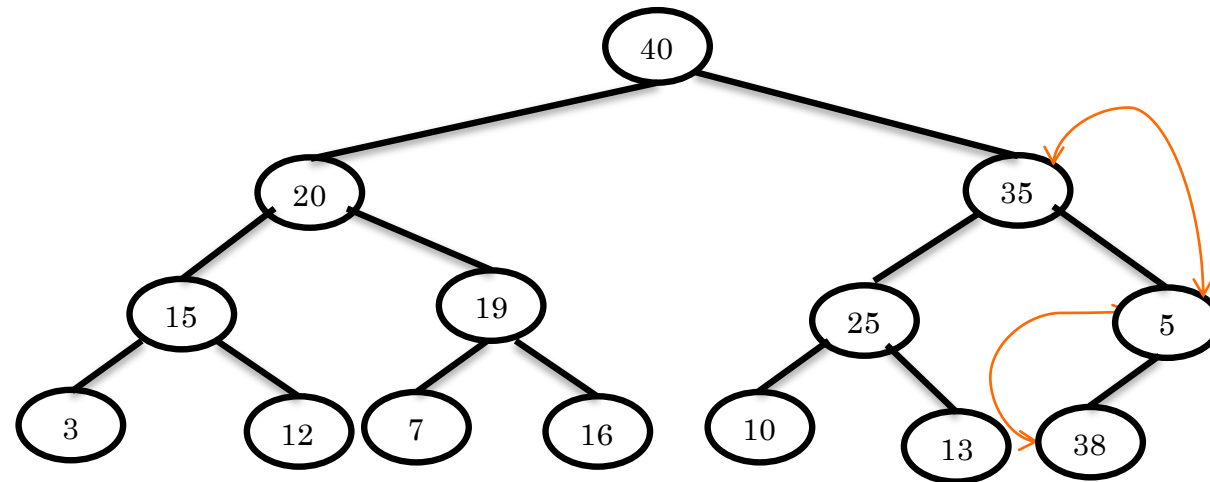
- ❖ **Solution:** Construire un $TAS_{\max}$ à partir des valeurs suivantes:



III.TAS: INSERTION

- ❖ **Solution:** Construire un $TAS_{\max}$ à partir des valeurs suivantes:

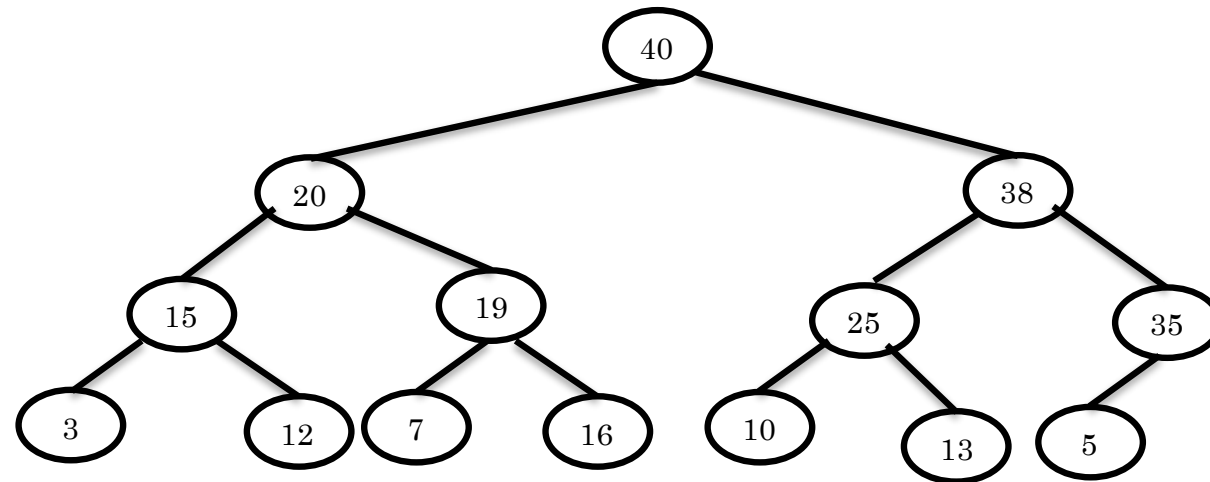
20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----



III.TAS: INSERTION

- ❖ **Solution:** Construire un $TAS_{\max}$ à partir des valeurs suivantes:

20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----



III.TAS: EXEMPLES D'APPLICATION

- ❖ Les TAS sont très utiles entre autre pour le **tri** et pour implémenter les **files de priorité**.

COURS ET METTRE EN TD)

TRI PAR TAS

- ❖ Étant donné un tableau d'entiers T (n : sa taille), dire comment peut on trier (ordre croissant) ce tableau en utilisant un $TAS_{\min}$ ou $TAS_{\max}$?

- ❖ **Exemple:**

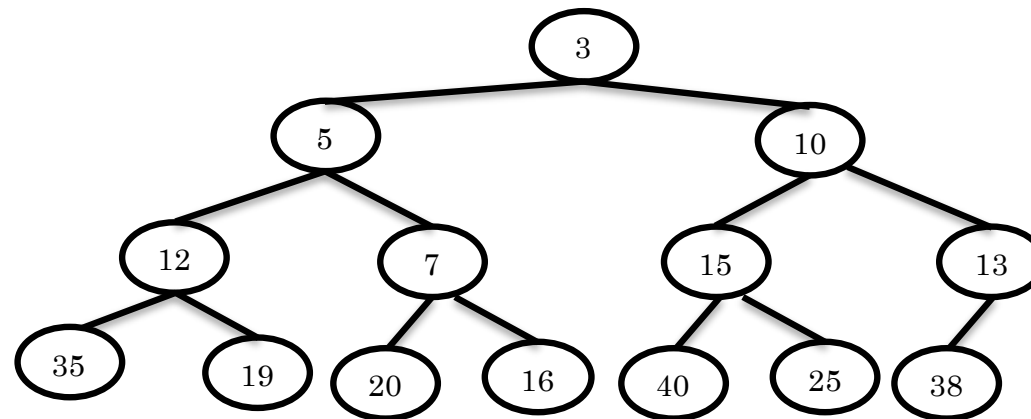
20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----

III.TAS: EXEMPLE D'APPLICATION

TRI PAR TAS

1. Transformer le tableau en un TAS_{MIN}

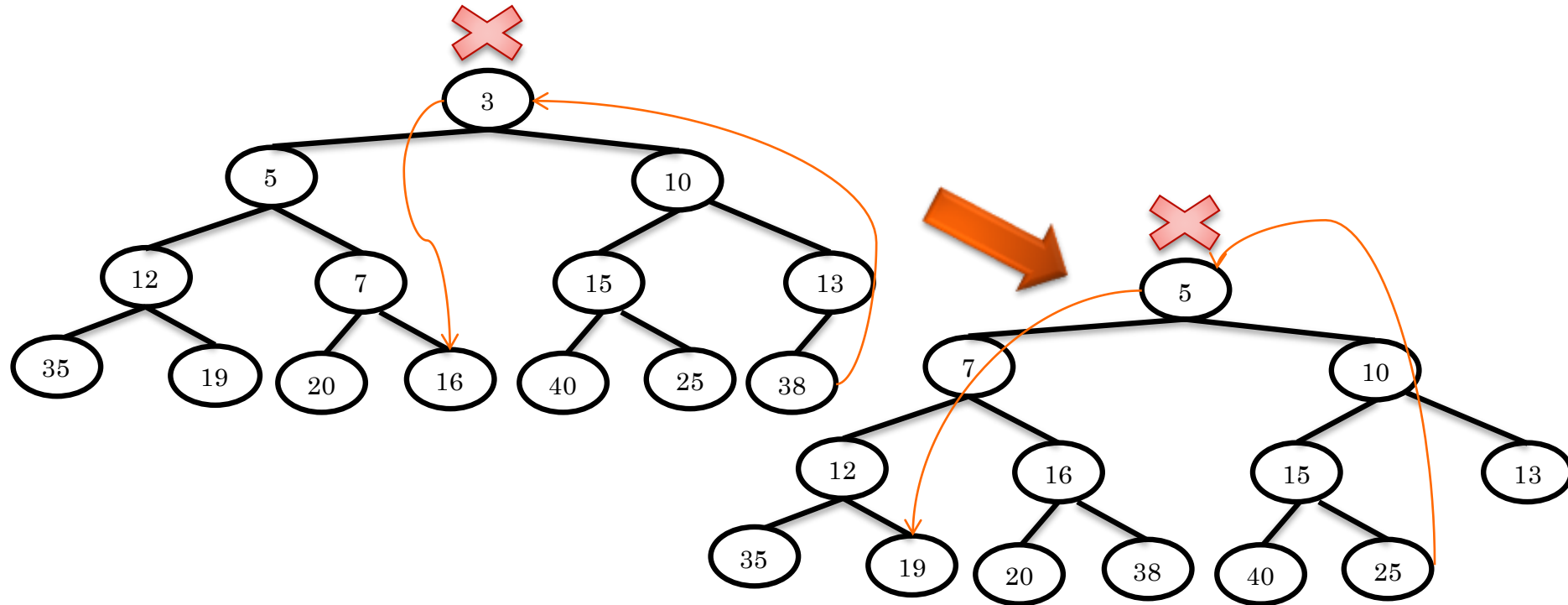
20	15	10	35	19	13	5	3	12	7	16	40	25	38
----	----	----	----	----	----	---	---	----	---	----	----	----	----



III.TAS: EXEMPLE D'APPLICATION

TRI PAR TAS

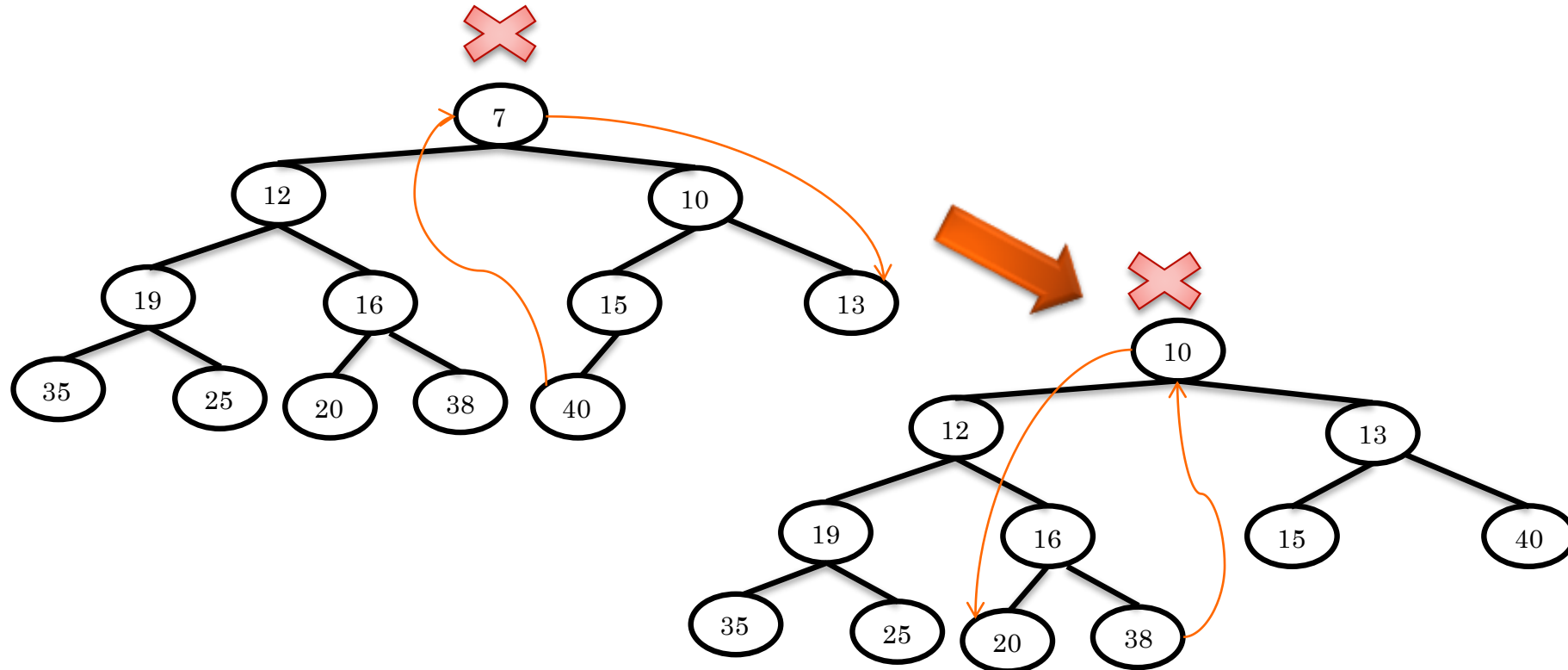
2. Extraire n fois le minimum du TAS_{MIN} :



III.TAS: EXEMPLE D'APPLICATION

TRI PAR TAS

2. Extraire n fois le minimum du TAS_{MIN} :

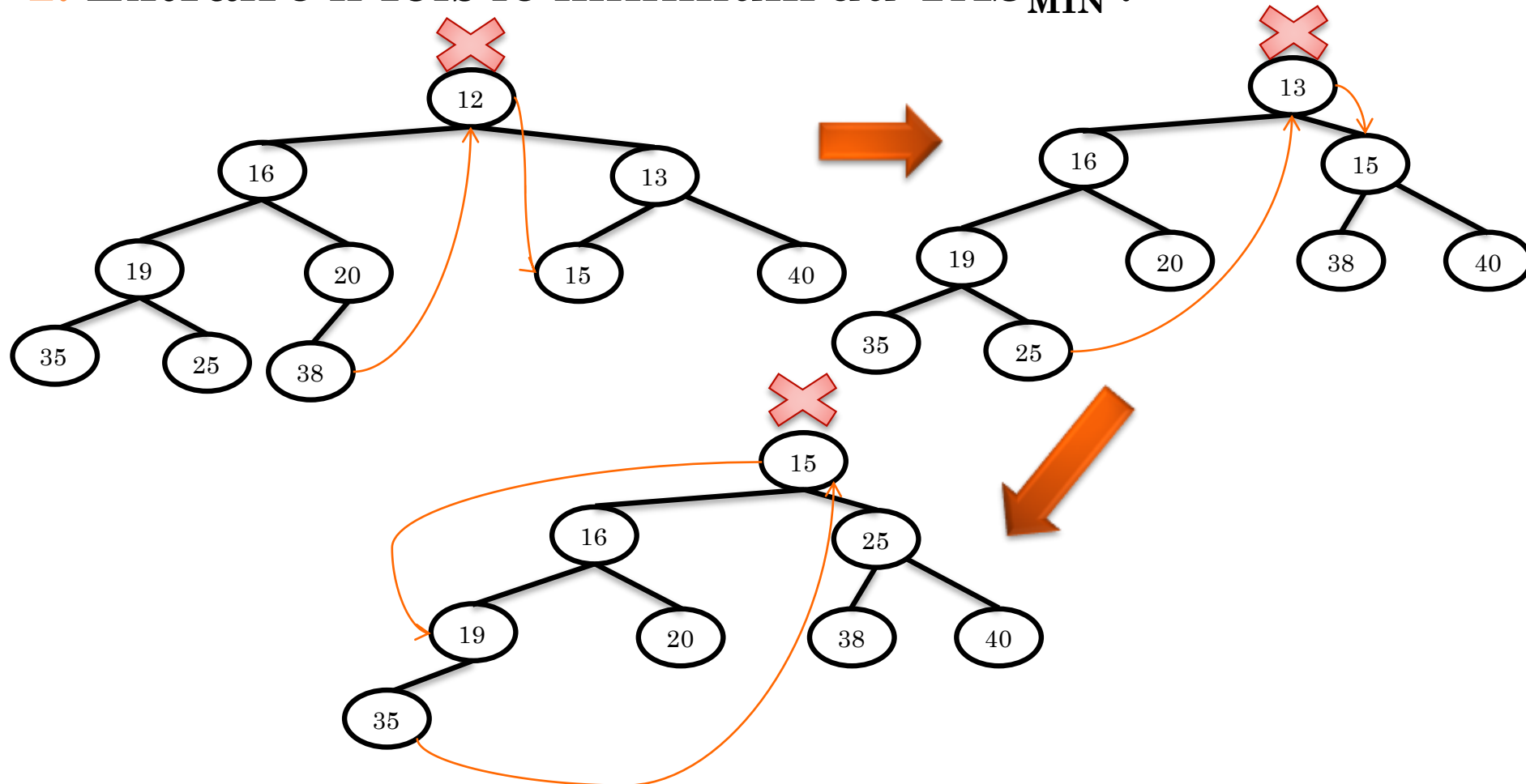


3	5
---	---

III.TAS: EXEMPLE D'APPLICATION

TRI PAR TAS

2. Extraire n fois le minimum du TAS_{MIN} :

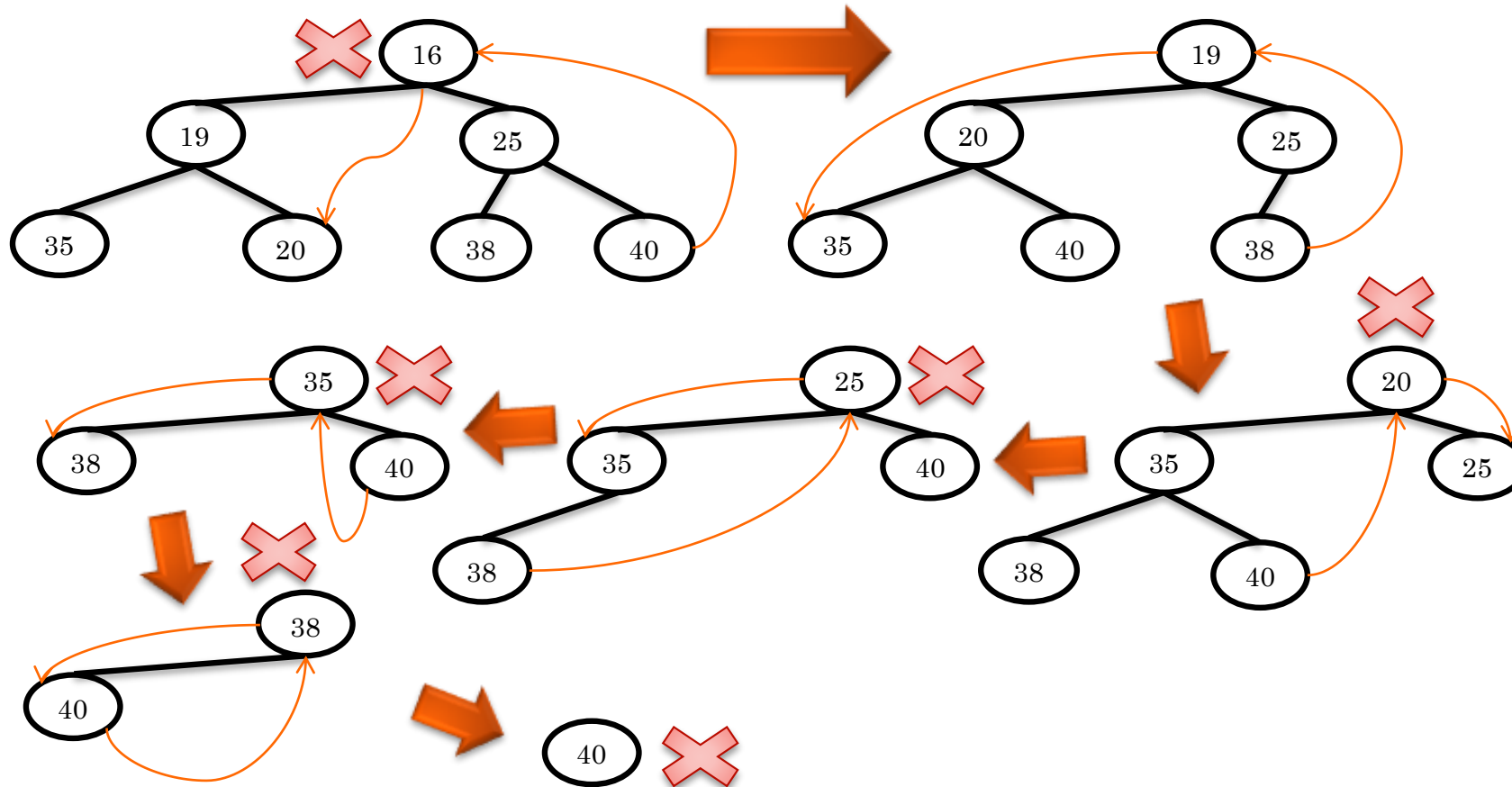


3	5	7	10
---	---	---	----

III.TAS: EXEMPLE D'APPLICATION

TRI PAR TAS

2. Extraire n fois le minimum du TAS_{MIN} :



3	5	7	10	12	13	15
---	---	---	----	----	----	----

III.TAS: EXEMPLE D'APPLICATION

IMPLÉMENTATION D'UNE FILE AVEC PRIORITÉ

- ❖ Une file d'attente avec priorité est une collection d'éléments dans laquelle l'ajout ne se fait pas toujours à la queue (fin).
 - Tout nouvel élément est ajouté dans la file, selon sa priorité.
 - Le retrait se fait toujours du début.
- ❖ Dans une file avec priorité, un élément prioritaire prendra la tête de la file même s'il arrive le dernier.
- ❖ L'implémentation de ces files d'attente peut être par tableau (décalage ou circulaire) ou LLC ordonnée, mais l'implémentation la plus efficace et la plus répandue utilise les **TAS**.